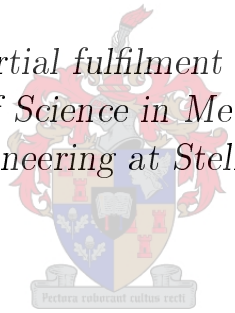
Control of a Conveyor System for a Reconfigurable Manufacturing Cell

by

Anro le Roux

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Mechanical Engineering in the Faculty of Engineering at Stellenbosch University



Department of Mechanical and Mechatronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. A.H. Basson

December 2013

…

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Abstract

Control of a Conveyor System for a Reconfigurable Manufacturing Cell

A. le Roux

Department of Mechanical and Mechatronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MScEng (Mechanical)

December 2013

This work entails a study of the control software of transportation systems for use in reconfigurable manufacturing systems (RMSs). Various control approaches are considered, with the focus on enhancing reconfigurability. The work is unique in the sense that the RMS is designed to manufacture small parts/products and is meant to be used in developing countries. Manufacturing systems that can ensure product quality and delivery, are a critical need in countries where the bulk of manufacturing systems function with manual labour. RMSs and holonic manufacturing systems (HMSs) are identified as concepts that can potentially compete with manual manufacturing systems. The competing system must thus have a low initial adoption risk, be able to adapt to changing product functionality and demands, and have a comparable throughput rate. IEC61311-3, IEC64199 function block and agent-based control architectures are evaluated. The control software is tested on an experimental conveyor system.

The thesis shows that IEC61131-3 and IEC64199 architectures are advantageous in lower levels of control. IEC64199 function blocks provide human interface and development tools and simplifies the distribution of control. The human interface and development tools of IEC64199 function blocks may prove beneficial in providing system monitoring and rapid low skilled adaptation of the control system, increasing reconfigurability of systems in under-developed countries. Unfortunately, the low maturity of the development environments for IEC64199 function blocks is a limitation. It is shown that an IEC64199 function block controller becomes complex as the actuator/sensor count exceed 10. Agent-based systems offer reliable control and powerful communication

tools but requires a higher level of expertise than IEC64199 function blocks. Agent-based systems are proposed for the core high level control. Complex systems can be controlled with agents and intelligence can be added to control systems in a reconfigurable way. For the reconfigurable control of large manufacturing systems, agent-based control was found to be superior to IEC64199 function blocks.

Uittreksel

Beheer van 'n Vervoer Band vir 'n Herkonfigureerbare Vervaardiging Sel

("Control of a Conveyor System for a Reconfigurable Manufacturing Cell")

A. le Roux

Departement Meganiese en Megatroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MScIng (Meganies)

Desember 2013

Hierdie werk behels 'n studie in die beheersagteware van vervoerstelsels vir die gebruik in herkonfigureerbare vervaardigingstelsels. Verskeie benaderings word oorweeg, met die fokus op die verbetering van herkonfigureerbaarheid. Die werk is uniek in die sin dat die herkonfigureerbare vervaardigingstelsel ontwerp is vir die vervaardiging van klein onderdele/produkte en is bedoel vir die gebruik in die ontwikkelende lande. Vervaardigingstelsels wat die kwaliteit van die produk en aflewering kan verseker, is 'n kritieke behoefte in die lande waar die grootste deel van die vervaardiging met handearbeid gedoen word. Herkonfigureerbare vervaardigingstelsels en holoniese vervaardigingstelsels is geïdentifiseer as konsepte wat moontlik kan meeding met die handmatige produksie-stelsels. Die mededingende stelsel moet dus 'n lae aanvanklike aannemingsrisiko hê, in staat wees om te kan aanpas by die veranderende produk funksionaliteit en aanvraag, en 'n vergelykbare deurvloeikoers kan lewer. IEC61311-3, IEC61499 funksie-blok en agent-gebaseerde beheer argitekture word geëvalueer. Die beheer sagteware is getoets op 'n eksperimentele vervoerband stelsel.

Die tesis toon dat IEC61131-3 en IEC61499 argitekture voordelig is in die laër vlakke van beheer. IEC61499 funksie-blokke voorsien menslike koppelvlak en ontwikkelings-gereedskap, en vereenvoudig die verspreiding van beheer. Die menslike koppelvlak en ontwikkelings-gereedskap van die IEC61499 funksie-blokke is moontlik voordelig in die voorsiening van stelselmonitering en vinnige laag-geskoolde aanpassing van die beheer stelsel. Dit mag dus moontlik die herkonfigureerbaarheid van stelsels, in onder-ontwikkelde lande, verhoog.

Die lae vlak van volwassenheid van die ontwikkelingsomgewings vir IEC61499 funksie-blokke verlaag hul bruikbaarheid. Daar word aangetoon dat IEC61499 funksie-blok beheerders baie kompleks raak as die hoeveelheid van aktueerders en sensors meer as 10 raak. Agent-gebaseerde stelsels bied betroubare beheer, en kragtige kommunikasie-gereedskap, maar vereis 'n hoër vlak van kundigheid as IEC61499 funksie-blokke. Agent-gebaseerde stelsels word voorgestel vir die hoëvlak beheer. Komplekse stelsels kan beheer word met agente en intelligensie kan bygevoeg word om stelsels te beheer in 'n herkonfigureerbare manier. Dit was gevind dat agent-gebaseerde beheer beter is as IEC61499 funksie-blok beheer vir die herkonfigureerbare beheer van groot vervaardigings stelsels.

Acknowledgements

It is a privilege to honour everyone who supported and enabled me to complete this work. I would like to express my sincere gratitude to the following people and organisations.

- Professor A.H. Basson for giving me this opportunity, for excellent guidance, highly professional support and much care for us, your students
- AMTS for financially supporting this research with the AMTS Grant AMTS-07-10-P
- CBI-Electric Low Voltage for supplying valuable study case information
- TIA and DST for taking interest in the project, visiting us in the Automation Lab and providing great advice
- The staff at the department of Mechanical and Mechatronic engineering at Stellenbosch University; Reynaldo Rodriguez, for your professional technical support; Mr. Cobus Zietsman, Mr. Ferdi Zietsman and the workshop staff, for your willingness to help; Mrs Welma Liebenberg and the administrative personal, for taking care of all the administrative tasks
- My fellow students and colleagues, Chibaye Mulubika and Karel Kruger, for the good times and late demo nights
- My family, for all the prayers
- My father, mother, sister, for always believing in me, always loving me and dedicating you lives to give me the very best, I could have never asked for a better home

Above all, I want to thank my father in heaven, for always being with me, for giving me this life. By Your grace, may I live it to the fullness of my abilities and glorify You with all I have.

Dedications

Hierdie tesis word opgedra aan my familie...
...en ons God.

When I look at your heavens,
the work of your fingers,
the moon and the stars,
which you have set in place,
what is man that you are mindful of him,
and the son of man that you care for him?
Yet you have made him a little lower than the heavenly beings
and crowned him with glory and honor.
You have given him dominion over the works of your hands;
you have put all things under his feet,
O Lord, our Lord,
how majestic is your name in all the earth!

Psalm 8

Contents

List of Figures

List of Tables

Nomenclature

Terminology

Manufacturing Cell: A part of a manufacturing system allocated for a specific manufacturing process. A manufacturing cell contains several sub-systems.

Cell Controller: The controller of the manufacturing cell.

Sub-Systems: Smaller manufacturing systems that works together to make up the manufacturing cell.

Computer: A standard desktop or laptop computer.

Operators: The personnel (humans) that are assigned to operate the system in the industrial environment. This may involve deployment, configuration, operation, maintenance and reconfigurations.

Intelligence: The autonomous reconfiguration intelligence or the ability for the system to autonomously adapt to reconfigurations (see chapter 3).

Human Effort (or just effort): The amount of work (effort) required by humans to adapt a system during reconfigurations (see chapter 3).

Model: An abstraction (model) of the physical transportation system in which all physical devices and their properties are represented. A Petri-Net is used for the model and contains positions and transitions. (see section 5.3.2).

Computer Aided Configuration Environment: An environment that assists the operator in creating the model and compile configuration files like the descriptor file and the configuration data block defined below (see section 5.3.3).

Interpreter: A program that establishes communication between the controller running on the PLC and the controller running on a computer (see section 5.3.4).

Descriptor File: A file that contains the descriptors that are loaded by the interpreter. The descriptors link PLC memory addresses to commands that can be exchanged with computers (see section 5.3.4).

Configuration Data Block: Data tables (that together makes the data block) that contain the system configuration information (see section 5.3.2).

Path Control: The control for selecting the path by which the pallets move through the conveyor system (a sequence of transitions, see section 5.3.2).

Traffic Control: The control for preventing collisions of pallets occur in the conveyor system.

Action Control: The control for activating the sequence of actuators to move a pallet from one position to the next (a single transition, see section 5.3.2).

Parameters

C_{asr}	Autonomous system reconfiguration cost	[unitcost]
C_{hyp}	Hyperbolic coefficient	[ ]
C_{msr}	Manual system reconfiguration cost	[unitcost]
R_N	Number of reconfigurations	[ ]

Variables and Functions

C_{dr}	Reconfiguration design cost function	[unitcost]
C_r	Reconfiguration cost function	[unitcost]
E	Human Effort variable	[%]
I	Intelligence variable	[%]

Abbreviations

ACL	Agent Content Language
AGV	Automated Guided Vehicle
AMTS	Advanced Manufacturing Technology Strategy
CBI	CBI-Electric Low Voltage
CNC	Computer Numerical Controlled
FMS	Flexible Manufacturing System
GUI	Graphical User Interface
HLC	High Level Controller
HMI	Human Machine Interface
HMS	Holonic Manufacturing System
IP	Internet Protocol
LLC	Low Level Controller
PLC	Programmable Logic Controller
RFID	Radio Frequency Identification
RMS	Reconfigurable Manufacturing System
RTS	Reconfigurable Transportation System
XML	Extensible Mark-up Language

Chapter 1

Introduction

The concept of Reconfigurable manufacturing systems (RMSs) was proposed by the University of Michigan in 1995 (Koren and Shpitalni, 2010). The committee on visionary manufacturing challenges identified RMSs as a priority technology for 2020 (see section 2.2). It is the opinion of the author and other experts that RMSs is the next generation of manufacturing systems.

1.1 Background

The Department of Mechanical and Mechatronic Engineering at Stellenbosch University started a RMSs investigation in 2006. The Advanced Manufacturing Technology Strategy, an initiative of the Department of Science and Technology of South Africa, had an 'affordable automation' theme in which a RMS project originated. Professor A.H. Basson at Stellenbosch University, together with Professor H. Vermaak of the Central University of Technology (CUT), formed part of one of three consortia under the RMS project. The RMS project was later transferred to the Technology Innovation Agency.

CBI-Electric Low Voltage (CBI) was identified as a possible case study for implementing the new technology developed in the RMS project. CBI has a wide range of products for which the production volumes range from high, for common models, to very low for the less common models. CBI thus offers an environment for which RMSs may prove to be valuable. A partnership with CBI developed, in which they offered generous case study information.

The local RMS project started with Sequeira (2009) and Dymond (2009) conducting feasibility studies in fixture-based and fixtureless approaches, respectively, for reconfigurable assembly systems. Various students worked on building the resulting experimental system at the Stellenbosch University Automation Lab (see figure 1.1). The reconfigurable assembly systems includes a pallet magazine designed by Burger (2009), a feeding sub-system (including a singulation unit designed by Strauss (2009)), an inspection and removal system, a conveyor sub-system and a welding sub-system.

A study on reconfigurable assembly system control, using the feeder and pallet magazine as case studies, was done by Adams (2010). Students at CUT started development of an agent-based cell controller for the reconfigurable assembly system. Du Preez (2011), an Industrial Engineering student at Stellenbosch University, developed a simulation procedure which can determine the performance of manufacturing systems as a function of system configuration and product mix. The work of Du Preez (2011) gave insight to sensible reconfigurations and is used to decide on the reconfiguration in section 4.1. A French exchange student, Poletti (2011), designed a second singulation unit.

Three students started their MSc studies in 2011, working on the individual sub-systems and their local integration, as well as external integration with CUT. Kruger (2011) worked on the control of the singulation unit designed by Poletti (2011) and a robotic feeding sub-system. The author worked on the control of the transportation sub-system. Mulubika (2013) worked on the control of the welding robot, as well as a local cell controller (apart from that of CUT).

A conveyor sub-system was installed together with an adjacent six degree of freedom robot (see section 4.1). The robot picks up parts from a singulation unit or part magazine and places the parts in a fixture, fitted on the pallet that runs on the conveyor. The robot, part magazine and the singulation unit represent the feeder sub-system. The pallet magazine and welding robot were installed at other stations on the conveyor. Together with CUT, a major full system demonstration was presented to the Technology Innovation Agency in July 2012 to demonstrate the system's functionality. Further academic research such as reconfiguration feasibility studies were also conducted on each student's responsible areas, as part of their thesis research.

1.2 Motivation

Manufacturing industries in developing countries are facing many problems with manual labour. Manual manufacturing systems are initially less expensive than automated systems, but less reliable and predictable. Such manual systems result in problems with product quality assurance and product delivery. Additionally, strengthening labour laws increase the cost of manual systems. Automated manufacturing systems are alternatives to manual systems that are much more deterministic. Automated systems are becoming more attractive when the costs of manual systems increase to levels comparable with that of automated systems.

On the other hand, the need for a manufacturing system to produce changing parts and varying production demands has become the norm (see section 2.1). Unlike manual systems, traditional automated systems (like dedicated manufacturing lines and FMSs described in section 2.2) do not have both adaptability and high production throughput rates in common. RMSs were

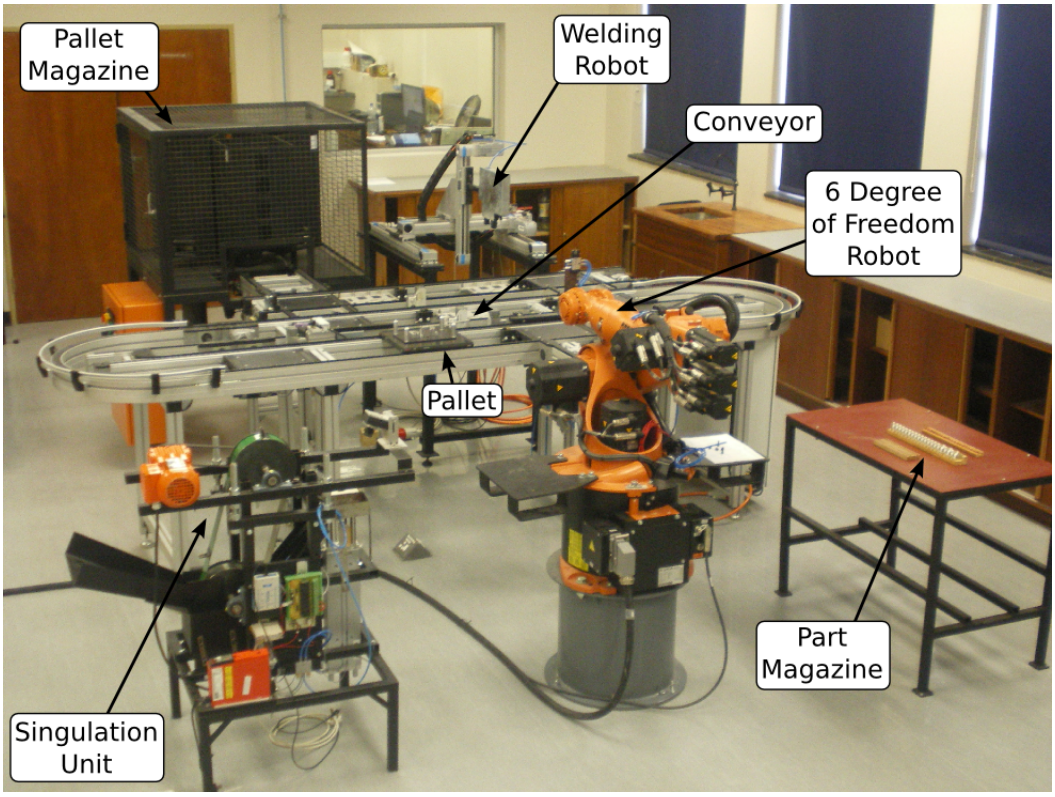


Figure 1.1: An experimental set-up of the manufacturing cell

introduced as manufacturing systems that combine the various strengths of traditional automated systems, and discard their weaknesses.

The design of RMS control is, however, still a major challenge. Despite the advantages of RMSs and the fifteen years of research that has gone into it, the industry is reluctant to adopt this technology. The author believes that the designing of RMSs for developing countries could lead to additional discoveries in the implementation of RMSs. Developing countries may incubate the perfect undiscovered conditions for RMSs. Such systems may, in-turn, present the industries of developed countries with RMSs that are less risky to adopt.

Transportation systems are a crucial part of manufacturing systems since they move parts and products between manufacturing stations (or resources). Conveyor systems are a commonly used for transportation in manufacturing systems and a modular conveyor was chosen as a study case for representing a reconfigurable transportation system (RTS). As the manufacturing system is reconfigured, the conveyor system must adapt to continually serve the resources in the system. The conveyor system must thus be reconfigurable on a hardware level, as well as a software level. The reconfiguration of the software level (the control system) of the conveyor system is the focus for this thesis.

1.3 Objectives

This study evaluates alternative strategies for the control of a modular conveyor system as a reconfigurable transportation system (RTS). The RTS will form part of a RMS that is being developed for under-developed countries. The control system must be reconfigurable, possessing the six key characteristics of RMS listed in section 2.3. The characteristics of the holonic manufacturing system (HMS) paradigm, also listed in section 2.3, will additionally be considered to enable the system design for under-developed countries.

Four control architectures are considered (see section 5.2). These control architectures are evaluated on the basis of their appropriateness for use in a RMS. IEC61131-3 is the industry standard for PLC control programs. IEC61499 function block control and agent-based control are regularly mentioned in RMS literature. For this reason these architectures are primarily tested and compared with each other. Object-Orientated control architectures are used for additional control. From the background studies in section 1.1, the use of non-reconfigurable fixtures is assumed for the holding of the parts/products. The fact that fixtures are fitted in a homogeneous transportation package, a pallet in this case, assumes that no product or part will exceed the dimensions of the pallet. This study only considers off-line reconfiguration, meaning that new configurations are built and simulated in a configuration environment (see section 5.3.3), decoupled from the controller, and downloaded onto the control hardware while the system is not running. The controller design does not include the configuration environment or high level intelligent modules like path planners and schedulers, but an interface to such intelligent modules is provided.

Different controllers are built using the various control architectures. These controllers are evaluated in a range of tests conducted on the conveyor system, to consider the performance in terms of reconfigurability. The conclusion is aimed at assisting the decision process for selecting appropriate control architectures or mixes of control architectures, for the conveyor system and even other RTSs.

Chapter 2

Literature Study

2.1 The Market Challenge

Koren *et al.* (1999) stated that economic competition is aggressive on a global scale, customers are more educated and demanding, and process technology is changing at a rapid pace. Koren *et al.* (1999) and Carpanzano and Jovane (2007) highlight factors that are becoming attainable and are affecting the markets, such as:

- spreading economic competition to a global scale,
- keeping customers alert, informed and educated,
- allowing product and process technology to develop at an increasing pace,
- allowing changes in parts for existing products,
- allowing a higher frequency of introduction of new products,
- the emerging mass customization and personalisation manufacturing paradigms and
- changes in governmental regulations, that are initiated by safety, environmental or political influences.

All these factors are mostly interdependent and have contradicting effects, making the prediction of product demands exceedingly complex. Manufacturers must find ways to adapt to these changes in a rapid and cost-effective manner. The rapid design and building or reconfiguration of a system, by itself, is not sufficient. A short ramp-up time (defined in section 2.3) is critical for successful adaptation to market changes.

Responsiveness provides a key competitive advantage in a turbulent global economy in which companies must be able to react to changes rapidly and cost-effectively Koren and Shpitalni (2010). Responsiveness refers to the speed at

which a plant can meet changing business goals and produce new product models. Responsiveness enables manufacturing systems to react rapidly and cost-effectively to:

- market changes, including changes in product demand,
- product changes, including changes in current products and introduction of new products and
- system failures (on-going production despite of equipment failures).

Although responsiveness has not yet been attributed the same level of importance as cost and quality, its impact is quickly becoming equally important.

2.2 The Development of Manufacturing Systems

Henry Ford's invention of the moving assembly line in 1913 marked the beginning of the mass production paradigm. Mass production was later only made possible through the invention of dedicated machining lines that produced the engines, transmissions and main components of automobiles Koren and Shpitalni (2010). Traditional manufacturing systems include dedicated manufacturing lines and flexible manufacturing systems (FMSs). Figure 2.1 illustrates the operating areas of these systems, as discussed in the following sections. Figure 2.2 indicates the characteristics that RMSs inherit from traditional systems. Holonic and bionic manufacturing systems are also discussed.

2.2.1 Traditional Manufacturing Systems

Dedicated manufacturing lines possess high production throughput rates and are cost effective only if they operate near full capacity (at point A or C in figure 2.1). If the production need is located at point B, two manufacturing lines will be needed and one of them will only run at 50% efficiency. The production capacity can only be scaled in big jumps and a high probability exist that one line will not operate at full capacity. Notice that dedicated manufacturing lines only operate in the region of the production of single parts/products.

Flexible manufacturing systems (FMSs) were developed to address the consumer needs for product variety. FMSs focus on the machine and have extended functionality to allow for the production of a general set of products. Basic FMSs, for example computer numerical controlled (CNC) machines, are successfully implemented in the industry. Figure 2.1 shows that FMSs are more scalable by adding machines in parallel. FMSs are designed to produce multiple parts, the opposite extreme to dedicated manufacturing lines.

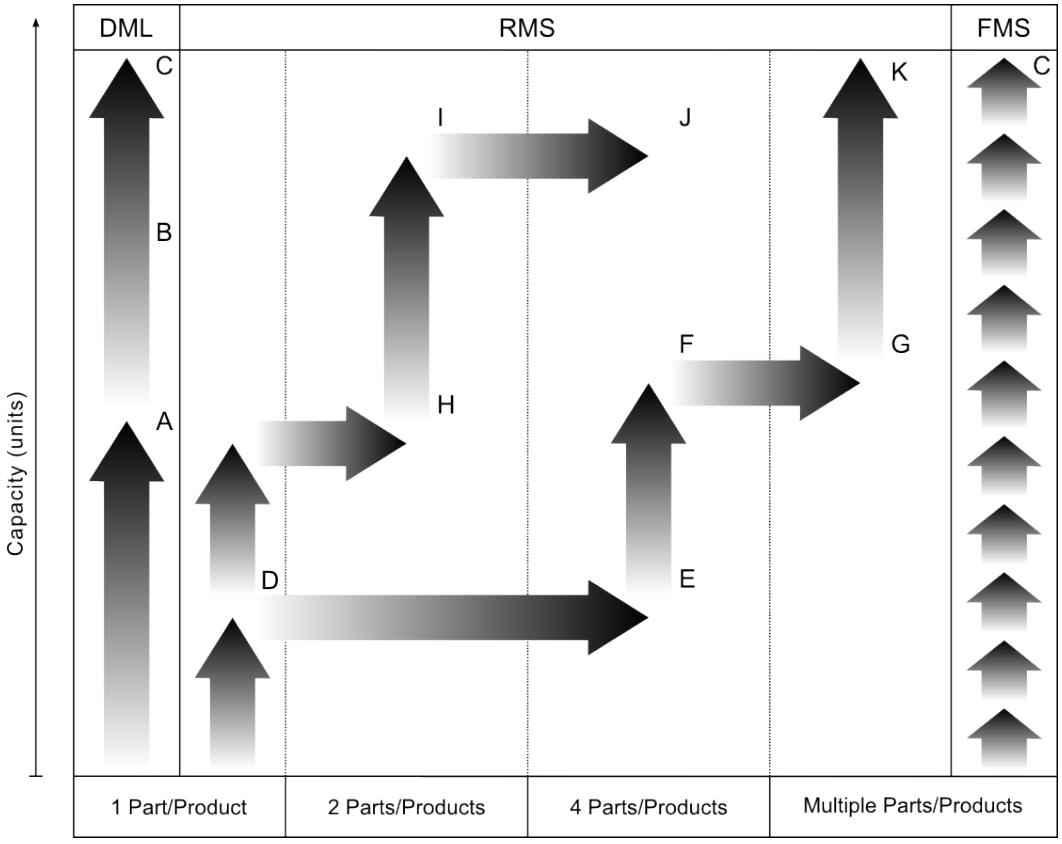


Figure 2.1: RMS operation regions (adapted from Koren *et al.* (1999))

	DML	RMS	FMS
System structure	Fixed	Changeable ←	Changeable
Machine structure	Fixed	Changeable	Fixed
System focus	Part →	Part family	Machine
Scalability	No	Yes ←	Yes
Flexibility	No	Customized ←	General
Simultaneous operating tools	Yes →	Possible	No
Productivity	Very high →	High	Low
Cost per part	Low →	Medium	Reasonable

Figure 2.2: RMS features inheritance (adapted from Koren and Shpitalni (2010))

While the FMS is mostly operating close to full capacity efficiency, it is operating at a low functionality efficiency. The reason for this is that FMSs are machine focused, possessing functionality that exceeds the requirement of the products to be produced. FMSs also have low production throughput rates and do not provide volume flexibility (Koren *et al.*, 1999; Koren and Shpitalni, 2010).

2.2.2 Reconfigurable Manufacturing Systems (RMSs)

RMSs were identified as a priority technology by the (Committee on Visionary Manufacturing Challenges, 1998). This identification was based on a workshop, a Delphi survey, briefings by technology experts and committee deliberations based on the following questions:

- Was the technology identified as a high priority technology in the Delphi survey?
- Was the technology identified as a high priority technology at the workshop?
- Is this a primary technology for meeting one of the grand challenges?
- Does the technology have the potential to have a profound impact on manufacturing?
- Does the technology support more than one grand challenge?
- Does the technology represent a long-term opportunity (i.e., is the technology not readily attainable in the short term)?

A RMS was introduced as a system that can be reconfigured to work close to its full capacity and functionality efficiency. Koren and Shpitalni (2010) define RMSs as:

> "Reconfigurable Manufacturing Systems are designed at the outset for rapid change in structure, as well as hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or regulatory requirements."

Figure 2.1 demonstrates how RMSs can be reconfigured to reach any production need points from D to K. RMSs inherit the good qualities from dedicated manufacturing lines and FMSs as shown in figure 2.2. Both the system and machine structures are changeable, allowing it to be scalable and customizable in terms of adaptability. The system focus of RMSs is on part families, meaning that the system is designed to be reconfigured for a specific number of parts/products. This property allows the system to be focused on the few

parts/products that are of concern, together with their specific production capacity, resulting in a high productivity and a medium part cost.

2.2.3 Holonic and Bionic Manufacturing Systems

The word "holon" was proposed over 30 years ago by Arthur Koestler (Koestler, 1969; Valckenaers *et al.*, 1998). It is a combination from the Greek "holos", meaning whole, with the suffix "-on", suggesting a part or particle such as a proton or neutron. Koestler proposed the concept of a holon, being impelled by the observations that:

- complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not and
- although it is easy to identify sub-wholes or parts, 'wholes' and 'parts' in an absolute sense do not exist anywhere.

The word holon describes the hybrid nature of sub-wholes/parts in real-life systems. Holons are simultaneously self-contained wholes to their subordinated parts, and dependent parts when seen from the inverse direction implying a hierarchical structure.

The field of holonic manufacturing was initiated in the early 1990s (Mcfarlane and Bussmann, 2000). It is intended to support highly responsive organizations by providing a modular building block or 'plug and play' capability for developing and operating a manufacturing production system. The Holonic Manufacturing Systems Consortium translated the concepts that Koestler developed into a set of appropriate concepts for manufacturing industries (Valckenaers *et al.*, 1998). The goal is to attain, in manufacturing:

- stability in the face of disturbances,
- adaptability and flexibility in the face of change and
- the efficient usage of available resources.

The HMS consortium developed the following list of definitions to help understand and guide the translation of holonic concepts into a manufacturing setting:

- Holon: An autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon.
- Autonomy: The capability of an entity to create and control the execution of its own plans and/or strategies.

- Cooperation: A process whereby a set of entities develops mutually acceptable plans and executes these plans.
- Holarchy: A system of holons that can cooperate to achieve a goal or objective. The holarchy defines the basic rules for cooperation of the holons and thereby limits their autonomy.

Bionic manufacturing systems attempt to use ideas from nature to overcome manufacturing challenges. Leitão *et al.* (2012) describe basic concepts of biology, swarm intelligence, evolution and self-organisation. Optimization evolutionary algorithms like ant colony optimization, the artificial bee colony algorithm, particle swarm optimization and genetic algorithms are used to solve mathematical engineering problems.

Leitão *et al.* (2012) also state that traditional approaches, based on centralized, rigid structures, do not have enough flexibility to cope with modularity, flexibility, robustness and reconfiguration For this reason, paradigms like multi-agent systems MASs, HMSs and bionic manufacturing systems have being introduced as distributed, autonomous and adaptive manufacturing systems.

2.3 Keys to Reconfiguration

Since the late 90s, technologies enabling reconfiguration have emerged (Koren *et al.*, 1999). Modular and open-architecture control software, together with modular machine tools, is being developed. Distributed artificial intelligence has been put in the spotlight with the increasing need to control widely distributed devices in disruption prone environments (Brennan, 2007). According to Brennan (2007), distributed artificial intelligence has led to the development of multi-agent systems (MASs), which has consequently led to the manufacturing specific application of holonic manufacturing systems. Considerable interest arose to extend research in HMS from a planning and scheduling level of control to the physical device level. The members of the HMS consortium focused on defining a low level control (LLC) architecture based on the IEC61499 function block standard. This work formed the basis for the use of IEC61499 in RMS. This means that HMSs is a subset of MASs and IEC61499 a LLC architecture under HMSs. From other literature is not clear if MASs and IEC61499 are both subsets of HMSs or if IEC61499 is a way of implementing HMSs and HMSs a subset of MASs like Brennan (2007) suggested. If Brennan (2007) is correct then IEC61499 is a definite LLC architecture while MASs is a HLC architecture.

A RMS must adhere to core characteristics as listed below. Customization, convertibility and scalability are critical RMS characteristics and guarantee modifications in product capacity and functionality. Modularity, integrability

and diagnosability allow for rapid reconfigurability. The six core characteristics of RMS are (Koren and Shpitalni, 2010):

- customization (the system is customized to be flexible only to a specific part family),
- convertibility (the ability to easily convert between producing different parts within the part family),
- scalability (the ability to easily modify production capacity),
- modularity (components are modular and can easily be rearranged),
- integrability (the ability to integrate components easily, e.g. standard interfaces or protocols) and
- diagnosability (the ability to detect, diagnose and correct the controller state or problems).

Originally the primary goal in developing reconfigurable manufacturing systems, was to develop machine modules, which can be quickly exchanged between different manufacturing systems (Koren *et al.*, 1999). This exchangeability can be accomplished by standardizing the interfaces combining the modules. Effective interfaces can rapidly decrease reconfiguration time. Various interface categories exist and must all be considered when designing RMS. The interface categories can each be either an international, national or company-specific standardized interfaces. Interfaces can be categorized into:

- system-interfaces (interfaces between process systems),
- module-interfaces (interfaces between modules in a system) and
- sub-module-interfaces (interfaces between sub-modules in a single module, e.g. a drive unit interfaced with a spindle result in a spindle module).

Each category can be further divided into:

- data interfaces,
- material transition interfaces,
- energy interfaces,
- mechanical interfaces which can be:
  - fixed (more permanent connections, e.g. drive bolted onto spindle) and
  - dynamic (connections that change often, e.g. grippers).

A short ramp-up time is another key to reconfiguration. Koren *et al.* (1999) defines the ramp-up period as:

> "The period of time it takes a newly introduced reconfigurable manufacturing system to reach sustainable, long-term levels of production in terms of throughput and part quality, considering the impact of equipment and labour on productivity."

The reduction of ramp-up time is critical for responsive manufacturing systems. The reconfiguration advantage is lost if ramp-up time is not kept short. Especially where the systems are reconfigured more often, it becomes essential to rapidly tune the system to high quality full production.

The modular nature of RMS and the various interfaces that these modules have, may lead to deteriorating quality, and makes it imperative to perform diagnostics, error calibration and compensation. Reconfigurable sensors, module sensors and measurement equipment are needed to detect problems. Systematic methodologies for root-cause analysis of part quality problems combined with rapid methods for on-line part inspection are keys for a rapid ramp-up. Consequently, reconfigurable systems must be designed with product quality measurement systems as an integral part.

The characteristics of HMSs assist the design of distributed systems, which is a key for RMSs. Christensen (1994) provided a good starting point for the basic requirements of next-generation manufacturing systems. Cost was added to these requirements and is listed by Brennan (2007) as:

- disturbance handling that provides better and faster recognition of and response to machine malfunctions, rush orders, unpredictable process yields, human errors, etc.,
- human integration supports better and more extensive use of human intelligence,
- availability provides higher reliability and maintainability despite system size and complexity,
- flexibility supports continuously changing product designs, product mixes, and small lot sizes,
- robustness maintains system operability in the face of large and small malfunctions and
- the cost of the initial development of the control system as well as the maintenance cost of the control system.

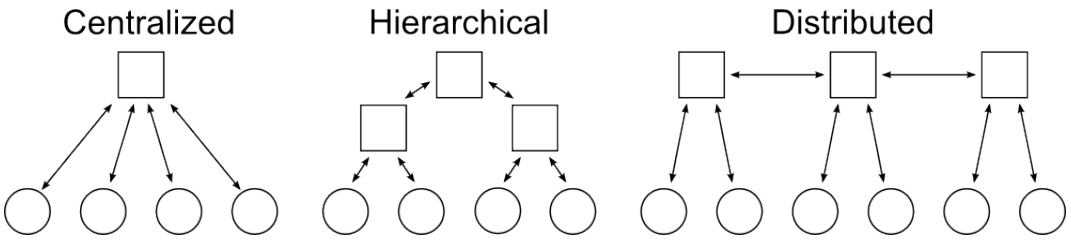


Figure 2.3: Control architectures (Meng *et al.*, 2006)

2.4 Reconfigurable Control

Meng *et al.* (2006) distinguishes between three kinds of control architectures (figure 2.3). Centralized control architectures have all the resources connected to one controller, while hierarchical architectures have levels of controllers, with higher levels controlling lower levels and/or resources. Distributed (or heterarchical (Brennan, 2007)) architectures use various controllers on the same level, communicating with each other and each controlling its resources. Meng *et al.* (2006) state that centralized and hierarchical modes have many shortcomings, such as structural rigidity, difficulty of control system design and a lack of adaptability. To add, modify or remove resources of centralized or hierarchical systems, the entire system must be shut down and all data structures of higher levels need to be updated, making reconfiguration expensive. In order to build distributed control systems, new tools and architectures are investigated.

2.4.1 Tools and Trends

This section discusses control modelling tools and control level design trends. An important aspect of reconfigurable systems is to keep the costs of reconfiguration as low as possible. The advances of system self-modelling are initially discussed. Further mathematical modelling tools, that can allow virtual system modelling, are also discussed as well as real-time reconfiguration and its issues. Finally, the use of high level and low level controllers are discussed.

In order to decrease the reconfiguration cost of control systems, system intelligence must increase to require less human effort to be reconfigured. Bongard *et al.* (2006) describe an active process that allows a machine to sustain performance through a process of self-modelling. The machine builds its own model by making actuations and measuring its response. Rather than building a new model from nothing (with no human input), it explores existing internal parametric models and finds the most fitting model to represent itself. It is possible for systems to autonomously synthesize increasingly complex behaviours through physical trial and error, but it requires hundreds or thousands of tests on the physical machine and is generally too slow, energetically costly and/or risky (Bongard *et al.*, 2006).

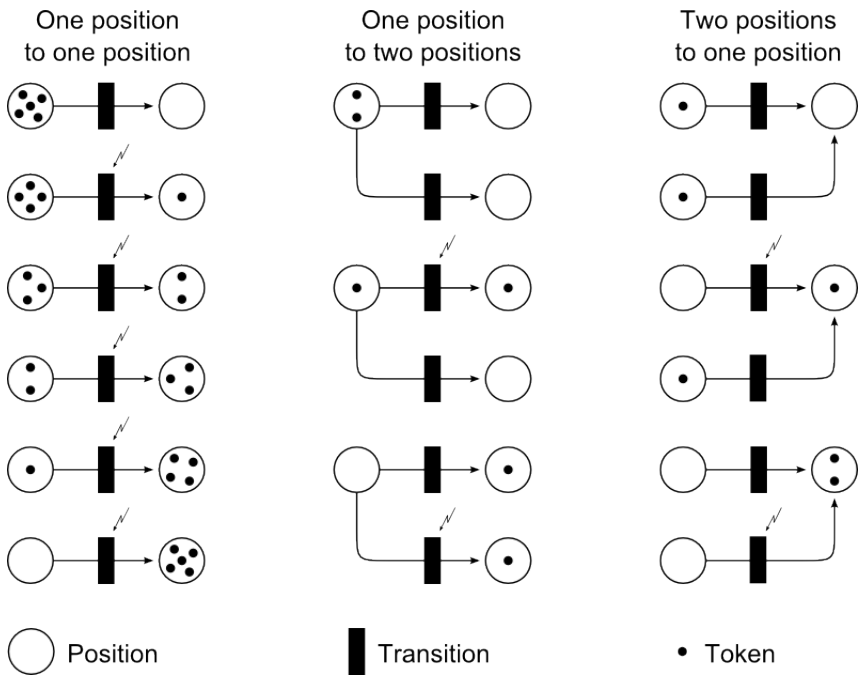


Figure 2.4: Petri-Net examples

The abundant use of discrete system mathematical models, for modelling RMSs, shows that such models are useful for manufacturing systems that need to be reconfigured. Finite state machines are regularly used to model states and state changes of low level controllers. Diego *et al.* (2010) have used finite state machines to model a flexible manufacturing plant with supervisory control. Petri-Nets are commonly used to model high level control of RMS (Wu and Zhou, 2011). A Petri-Net has positions that are connected with each other with transitions (see figure 2.4). The positions contain tokens that are transferred to other positions when the connecting transitions are triggered. The powerful analysis tools that Petri-Nets offer (Murata, 1989) can be used to solve complex high level problems. Mathematical models can act as a representation of real systems (like manufacturing plants (Diego *et al.*, 2010)) in order to hide complexity and diversity and allow system changes (Wu and Zhou, 2011).

Brennan (2007) looks at Christensen's (Christensen, 1994) requirements of disturbance handling and robustness. He states that automatic reconfiguration is key to an industrial system that is to quickly respond to change while maintaining stable system operation. He continues to say that reconfiguration in conventional PLC systems involves a process of first editing the control software off-line while the system is running, then committing the change to the running control program. When the change is committed to the running system, severe disruptions and instability can occur.

Brennan (2007) lists the barriers to widespread adoption of distributed real-time intelligent control systems as the:

- lack of skills in distributed system design,
- lack of a methodology to predict the aggregate behaviour of agents,
- cost of adoption and implementation,
- lack of mature global standards,
- lack of real-time agent development tools,
- lack of methodologies for verifications and validation and
- the need for formal work on safety-critical system design.

Two-level control approaches were suggested by Vrba *et al.* (2011) and Lepuschitz *et al.* (2011). Vrba *et al.* (2011) describe systems with the low level controller (LLC) running on a PLC and the high level controller (HLC) on a computer, as the usual approach. Controllers consist of two levels with a major reason being the different real-time requirements of the two layers (Lepuschitz *et al.*, 2011). A LLC is built on top of, and communicates directly with, the hardware level and can provide basic functionality without the HLC. The HLC is built on the LLC and enables intelligent decision making. According to Lepuschitz *et al.* (2011), the responsibilities of LLCs are:

- safe operation (in the case that the HLC is failing, the LLC should be able to work properly or reject the HLC suggestions that endanger proper system function),
- real-time execution,
- diagnosis (the detection of local events or disturbances in the physical system like direct sensor and hardware failures) and
- interaction (designers, operators, and maintenance personnel can directly access the LLC over defined interfaces).

IEC61131-3 standard PLCs are highly popular and were still the preferred low level controller in 2011, at Rockwell Automation (Vrba *et al.*, 2011). Brennan (2007) states that PLCs are fast, durable, reliable and predictable. He continues to say that they are cost effective in industrial environments. IEC61131-3 is the popular PLC programming language standard but does not support reconfigurable control. Vrba *et al.* (2011) suggests a method where IEC61131-3 code is generated (prior to control run-time) and downloaded into the PLC (refer to the code generation and download method in section 2.4.3). The external modelling and download method is used in this work (section 2.4.3 and 10.1.1) and is based on the code generation and download method of Vrba *et al.* (2011) (see section 5.3.3).

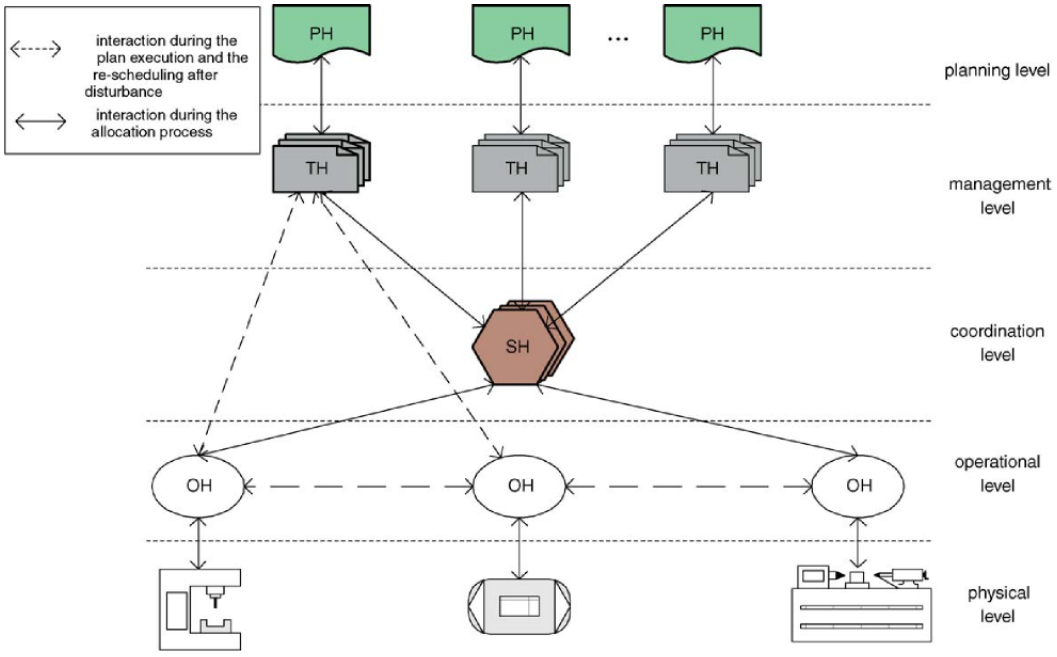
2.4.2 Holonic Reference Architectures

When developing HMSs, the ways in which holons can be used in control systems are endless. Reference architectures are used as frameworks to specify the types of holons and their tasks. Existing reference architectures are shown in table 2.1. PROSA is short for Product-Resource-Order-Staff Architecture and describes the use of product, resource, order and staff holons (Van Brussel *et al.*, 1998). The product holons hold the product and process knowledge. Order holons manages the tasks to be completed in the production process. Resource holons represent hardware devices and handle their management. Staff holons are added to assists the other three holons. PROSA is a well-known architecture that has served as a foundation for most holonic control architectures at the HLC level (Brennan, 2007).

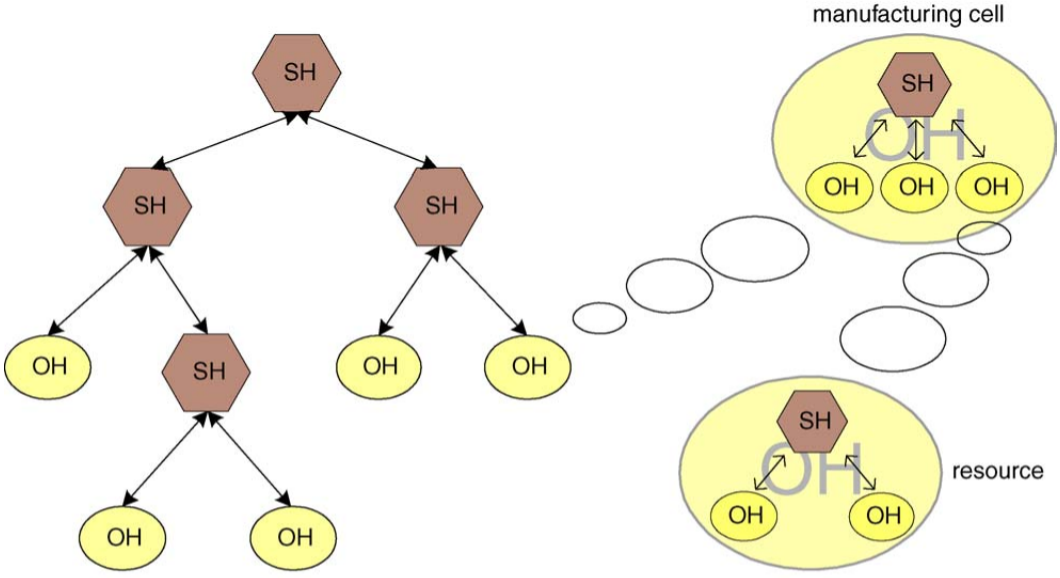
Leitão and Restivo (2006) proposed another control architecture, designated ADAptive holonic COntrol aRchitecture (ADACOR) for distributed manufacturing systems. The ADACOR architecture is based on the HMS paradigm and founded on decentralised systems, supervisor entities and self-organisation. The manufacturing control functions are in charge of a community of autonomous and cooperative holons. This function enables modularity, decentralisation, agility, flexibility, robustness and scalability. ADACOR defines four manufacturing holon classes namely: product holon, task holon, operational and supervisor holon. The product, task and operational holons are similar to the product, order and resource holons defined in the PROSA reference architecture. The introduction of the supervisor holons allow the establishment of hierarchies in decentralised systems, to achieve global production optimisation. As seen in figure 2.5(a), the supervisor holon communicates with task holons and operational holons. The idea is that the supervisor holon initially uses data from the task holons to control the operational holons, after which the task holons eventually bypass the supervisor holon and controls the task holons directly. The task holons receive product information from the product holons. The operational holons can also communicate among each other. Figure 2.5(b) shows how the holonic architecture is constructed of sub-holarchies.

2.4.3 IEC61131-3

IEC61131-3 is a programming language standard that is commonly used on PLCs and includes the popular ladder logic language. A major limitation of IEC61131-3 languages is that code (objects or classes) cannot be dynamically instantiated (at runtime) on the PLC. Rockwell Automation uses code generation and download method to reconfigure IEC61131-3 objects. Instances of holonic templates are created during the code generation and that LLC parts are assembled according to object-orientated enhancements of IEC61131-3 (Vrba *et al.*, 2011). These enhancements enable IEC61131-3 to create a



(a) ADACOR holonic reference architecture



(b) ADACOR sub-holarchies

Figure 2.5: ADACOR reference architecture (Leitão and Restivo, 2006)

Table 2.1: Holonic reference architectures (Vrba *et al.*, 2011)

Architecture	Order	Product type	Product instance	Resource/ Machine	Supervisor
PROSA		product holon	order holon	resource holon	staff holon
ADACOR		product holon	task holon	operational holon	supervisor holon
PABADIS	manufacturing order agent		product agent	residential agent	plant management agent
HCBA	coordinator	product holon	work-in-progress agent	resource holon	
KASA	order agent		supply agent	machine agent	contact agent
Rockwell	order agent	production plan agent	product agent	work station/ equipment agent	

holonic agent template with the object-orientated design methodology. These object-orientated enhancements are:

- indirect references (a technique to access attributes of the LLC templates),
- macro instructions (provide the system developer with the possibility to specify basic operations over a collection of components),
- inheritance (enables reuse of the existing object template's LLC part to define a new, more specific object class extending the original one) and
- containment (allows a designer to specify that a component (e.g. a conveyor) contains subcomponent/s (e.g. a motor), and then, traverse this containment tree arbitrarily).

The IEC61131-3 standard supports the use of ladder logic, function block or statement list language programming (Brennan, 2007). The statement list language is a low level assembler type language, while ladder logic and function block code are higher level programming methods. Ladder logic and function block code can be converted to statement list language since it is built on the

statement list language. Ladder logic present itself as relay logic diagrams. It is intuitive and very popular in the automation industry.

Lepuschitz *et al.* (2011) describe problems with IEC61131-3 function blocks that disables modularity and encapsulation:

- Global variables have the drawback that function blocks that seem decoupled and completely independent are linked together via a hidden interface.
- The IEC61131-3 applies a data-driven approach. That means that the data connections between the function blocks define the overall execution order. The user or a reconfiguration coordinator has no direct control of the execution order. Therefore, in the current IEC61131-3 systems, whole applications are changed at once and not gradually, as it would be needed for the automation holon.
- No unified (re)configuration interface exists that allows reconfiguration coordinators (e.g. a software agent) to change the application, a need when using a heterogeneous system with devices from different vendors, as it is typically applied in industrial automation.

2.4.4 IEC61499

The origin of IEC61499 is discussed in section 2.3. The IEC61499 standard is based on the IEC61311-3 standard, but allows the use of function blocks in distributed automation and is particularly relevant to the LLC level (Brennan, 2007). Vyatkin (2007) state that distributed automation will enable machines to be intelligent and implement a custom set of functions by means of adding and removing software components (IEC61499 function blocks). The custom set of functions may include:

- system simulation and modelling,
- programmability,
- service provision mechanism and
- the ability to be integrated with other intelligent machines in a seamless way.

According to Vyatkin (2007), IEC61499 provides a number of key technologies for encapsulation, portability, interoperability and configurability of heterogeneous distributed automation systems. These technologies are:

1. An open function block model provides a mechanism for encapsulation of automation knowledge and for making it portable.

2. Composite function blocks are platform independent thanks to the event connections between the component blocks.

3. An abstract device model provides a mechanism for creating new device types as a set of resource types and function block libraries.

4. The device management mechanism provides configuration of compliant devices by compliant software tools.

5. Service interface function blocks provide the mechanism for encapsulation of hardware creation software.

6. Part 2 of the standard defines the XML-based data format of function blocks that provides compatibility also on the level of source files.

7. Specific protocols of device configuration or management may be defined as compliance profiles to the standard.

Each function block has event inputs and outputs, as well as data inputs and outputs. Basic function blocks have internal variables, algorithms that use the internal and external (data in/output) variables and an execution control chart to schedule the execution of the algorithms using the event in/outputs. Function block networks are created by connecting the event in/outputs and data in/outputs of function blocks to other function blocks in the network. Composite function blocks can contain function block networks and present it as a single function block where the incoming and outgoing event and data lines are the event and data in/outputs. Device models represent actual devices in the system and each device contains resources. Function block networks are contained in the devices and resources. Each device has process and communication interface/s for communicating with other devices (Vyatkin, 2007).

Vyatkin (2007) also introduced the Model-View-Controller (MVC) design methodology. The MVC methodology aims at a function block implementation from the MVC object-orientated design pattern (Gamma *et al.*, 1994). First a sketch of the machine or process is made that describes its layout, degrees of freedom and its production cycle. View elements are created that shows a presentation of the manufacturing system. HMI elements are created with which the user can control the appearance of the view elements and animate them. The HMI elements fit in a HMI layer, while the view elements reside in a view layer. The HMI layer groups elements that are used by the human operator to control the system. The view layer groups view elements that function as a graphical display of the system.

A model layer is then created between the HMI and view layers. The model layer groups elements that model the behaviour of the manufacturing system. With the model elements handling the motion of the view elements, the HMI elements are changed to control the model elements. A control layer is added between the HMI layer and the model layer. The control layer groups

controller elements that can control the model elements. The HMI elements are further simplified to only interact with the controller elements.

Diagnostic elements are also added in the control layer and output information, from model elements, to HMI elements. Distribution is performed by allocating model elements to separate devices and using distributed links such as PUBLISH and SUBSCRIBE to replace local communication lines. The final design step is designing the controller to control the physical devices. This is done by replacing the model layer with an interface layer containing interface elements. The view layer is replaced with the machine or process layer containing the actual machine elements. The interface elements run on the distributed devices and communicate with the controller elements.

2.4.5 Object-Oriented Control

The simulation language Simula, developed in Scandinavia in the 1960s, is usually considered to be the original object-oriented programming language (Butler and Corbin, 1991). According to Butler and Corbin (1991), an object is a collection of data, together with all the operations which access or alter that data. The set of operations defined for an object constitute a uniform external interface to other parts of the system. Interaction with an object occurs only through requests for the object to execute one of the operations in its interface. The four principal concepts associated with an object-oriented style of programming are:

- encapsulation, which refers to the packaging of the object data and operations together in a single module to form an object,
- data abstraction, that is achieved when a collection of data and operations can be referred to by name (a class) and used to construct objects instances,
- inheritance that is a mechanism to enable the creation of new object classes which are extensions of other classes and
- dynamic binding that can be thought of as a facility for generalising procedure calls.

Zhang *et al.* (1999) highlight some important modelling methodologies and suggest the use of the object-oriented methodology as a better option. He continues to propose an object-oriented modelling methodology which consists of a physical manufacturing entity object, information manufacturing entity object and a control manufacturing entity object that can communicate with each other. A physical manufacturing entity object is an object with a tangible correspondent in the real world system. Physical manufacturing entity objects are components pertaining to material flow. Information manufacturing entity objects are objects which may or may not have a tangible correspondent in the

real world system. Information manufacturing entity objects are components with regard to information flow and process or store information in the manufacturing cell. A control manufacturing entity object is a conceptual object which typically has no tangible correspondent in the real system. The primary functions of the control manufacturing entity object are to evaluate the state of a given system, exercise a logic algorithm, and signal an appropriate action to be taken.

2.4.6 Agent-based Control

Bellifemine *et al.* (2007) states that agents are considered one of the most important paradigms that may improve on current methods for conceptualizing, designing and implementing software systems. Bellifemine *et al.* (2007) provides the following definition:

> "An agent is essentially a special software component that has autonomy and that provides an interoperable interface to an arbitrary system and behaves like a human agent, working for some clients in pursuit of its own agenda."

An agent is:

- autonomous, because it operates without the direct intervention of humans or others and has control over its actions and internal state,
- social, because it cooperates with humans or other agents in order to achieve its tasks,
- reactive, because it perceives its environment and responds in a timely fashion to changes that occur in the environment and
- proactive, because it does not simply act in response to its environment, but is able to exhibit goal-directed behaviour by taking initiative.

Moreover, if necessary an agent can be:

- mobile, with the ability to travel between different nodes in a computer network,
- truthful, providing the certainty that it will not deliberately communicate false information,
- benevolent, always trying to perform what is asked of it,
- rational, always acting in order to achieve its goals and never to prevent its goals being achieved and

- teachable, taught to adapt itself to fit its environment and to the desires of its users.

Initial efforts in the field of agent-based computing focused on the development of intelligent agent architectures, and the early years established several lasting styles of architecture. These range from purely reactive (or behavioural) architectures that operate in a simple stimulus-response fashion to more deliberative architectures that reason about their actions. Agent architectures are divided into four main groups: logic based, reactive, BDI (belief, desire, intention) and layered architectures. One of the key components of multi-agent systems is communication. Agents need to be able to communicate with users, with system resources and with each other if they need to cooperate, collaborate, negotiate and so on. The Foundation for Intelligent Physical Agents (FIPA) was established in 1996 as an international non-profit association to develop a collection of standards relating to software agent technology. The FIPA standard specifies an agent platform which holds an agent management system, a directory facilitator and any other agents that are created.

Leitão and Restivo (2006) used multi-agent technology to implement ADACOR on a productions control system prototype. The DaimlerChrysler's Production 2000+ project resulted in a system of modules that each consist of a CNC machine and a transportation system with three conveyors (Vrba *et al.*, 2011). The modules have overlapping capabilities and are connected via a redundant network of conveyors. Products are represented as agents that negotiate with machine module agents for the execution of operations. The system was installed in a real factory as part of a manufacturing line. Rockwell Automation also used agents in a steel mill application, a shipboard automation application and a material-handling application (Vrba *et al.*, 2011).

Chapter 3

Autonomous Reconfiguration Intelligence

Before a manufacturing cell is designed to be reconfigurable, it is important to know whether the added costs will be justified. For a system that will only be reconfigured twice in its life, a RMS may not be worth the additional design and implementation costs. The method presented in this chapter is therefore aimed at providing a design guideline to answer an important question: *Given the number of reconfigurations desired by the client, how much resources should be invested into designing and building the ability into the system to autonomously reconfigure?*

The term "autonomous reconfiguration intelligence", or just "intelligence" for brevity, is introduced in this chapter. This term is used to indicate the ability of a controller to reconfigure itself without human intervention or input. The method is aimed at estimating the target intelligence for a controller and requires four input parameters that can be estimated at design time. Although it is hard to measure intelligence, it is the opinion of the author that the four unknown input parameters can confidently be estimated by experts in the industry.

Without a design goal, the system may be either under-designed, requiring additional human effort for each reconfiguration, or over-designed, resulting in investments made to add intelligence that are not used. The primary focus in this section is on the design and reconfiguration of the controller. To illustrate the method's use in this chapter, typical values are used from the case study implementation described later in the thesis (see section 4.1).

3.1 Intelligence

Autonomous reconfiguration intelligence, further referred to as intelligence, is defined as follows: *Intelligence is a property that distinguishes the extent to which the control system can automatically handle the tasks of reconfiguration.*

Software units, ranging from a simple calculator or effective human interface environments (like IDEs or GUIs) to highly advanced artificial intelligent algorithms, are all components that increase intelligence. The intelligence scale ranges from 0% to 100%, where 0% is the entirely manual control or manual reconfiguration of control software, and 100% is fully autonomous controller reconfiguration. Intelligence is the major design parameter that needs to be determined by the designer, before the design of the controller can commence. Inversely proportional to the intelligence scale, the human effort scale (further referred to as effort) also ranges from 0% to 100% (effort = 100% where intelligence = 0%, and vice versa).

3.2 The Intelligence-Effort Relationship

Small additions of computer intelligence can dramatically reduce human effort when the control system has little intelligence. This statement can be supported by the many simple (low intelligence) computer programs (e.g. word processors) that are widely used in offices because of way they increase productivity. This observation suggests the hyperbolic intelligence-effort relationship in section 5.3.1, over a linear relationship (in figure 5.3).

Additionally, an assumption can also be made that, for an initial intelligence of 100% and a small addition of effort, intelligence can be dramatically decreased, as supported by the following supposition. Suppose a system is able to fully reconfigure itself (after a human arranged and connected all hardware) by executing randomized actuations with its actuators (outputs), measuring hardware activity with sensors (inputs) and discover its own configuration (discussed by Bongard *et al.* (2006)). Such a system (at point E in figure 3.1) must make use of state of the art machine learning technology to slowly discover things that an operator can easily see and suggest to the system with little effort.

Thus, the inverse proportionality between intelligence and effort is assumed to rather be a shifted hyperbolic function as described by equation 3.2.1,

$$I(E) = \frac{C_{hyp}}{E + s} - s \tag{3.2.1}$$

where I and E are the intelligence and effort percentages respectively, C_{hyp} is the hyperbolic constant and $s = \frac{1}{2}(\sqrt{1 - 4C_{hyp}} - 1)$. Equation 3.2.2 indicates the reconfiguration cost function (C_r), of equation 3.2.1, projected on the manual-auto reconfiguration cost surface (figure 3.1),

$$C_r(I, E) = C_{asr}I + C_{msr}E \tag{3.2.2}$$

with C_{asr} the fully autonomous system reconfiguration cost and C_{msr} the purely manual system reconfiguration costs. In equation 3.2.1, I and E can be interchanged due to symmetry about the $I = E$ line. By eliminating E

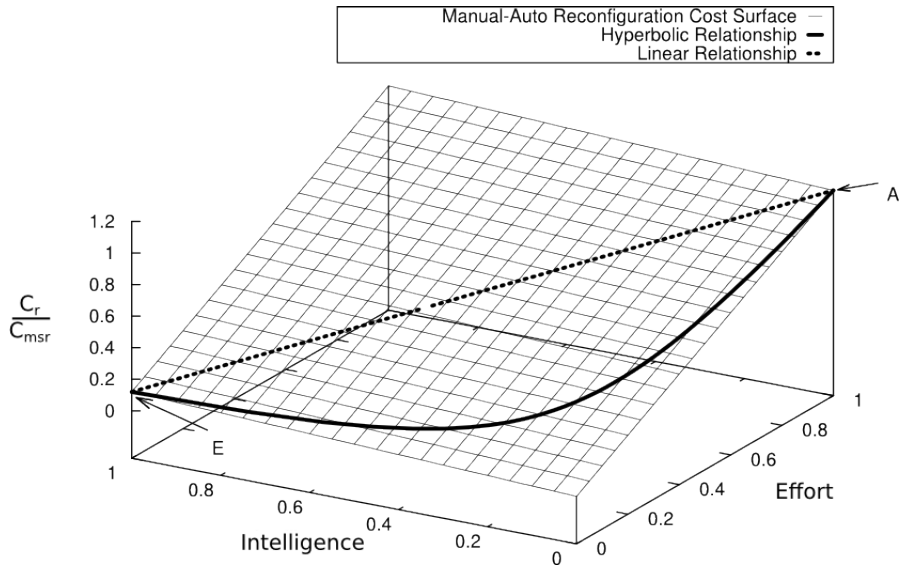


Figure 3.1: Intelligence-Effort graphs on the manual-auto reconfiguration cost surface

in equation 3.2.2, and using equation 3.2.1, equation 3.2.3 can be obtained to give C_r as a function of I.

$$C_r(I) = C_{asr}I + C_{msr}\left(\frac{C_{hyp}}{I+s} - s\right) \tag{3.2.3}$$

3.3 Reconfiguration Cost

The intelligence-effort relationship graph showed in figure 3.1 plots the manual-automation reconfiguration cost surface and the intelligence-effort curves, as a function of intelligence and human effort. The dotted line is a linear relationship where $C_{hyp} \to \infty$ and the solid line is a hyperbolic relationship with $C_{hyp} = 0.0695$. The first input parameter required by the model is therefore C_{hyp}. Whether the intelligence-effort relationship is linear ($C_{hyp} \to \infty$) or more hyperbolic, is highly debatable and can be obtained from statistical data and industry experience. Assuming that a 30% increase in intelligence will typically reduce the reconfiguration time by 87.5%, a hyperbolic constant of $C_{hyp} = 0.0695$ can be calculated.

Two other parameters, required to produce the manual-automation reconfiguration cost surface, are the autonomous and manual system reconfiguration costs (C_{msr} and C_{asr}). With large and complex systems, where the hardware (with inertia) must be actuated to get feedback from sensors, the fully autonomous reconfiguration process is generally too time consuming and dangerous (Bongard *et al.*, 2006). Nevertheless, a complete control software redesign by humans is predicted to be much more expensive than that of autonomous

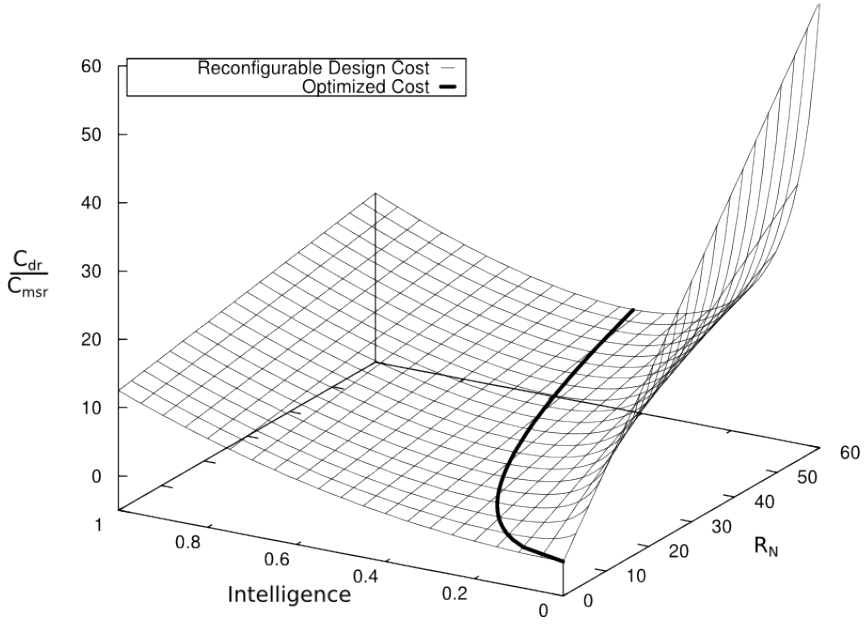


Figure 3.2: The controller intelligence level design investment graph

reconfiguration. The typical, $C_{asr} = 12000$ and $C_{msr} = 120000$ cost units, are used.

3.4 Optimal Intelligence Level

The optimal level of intelligence must be determined for the desired number of configurations to minimise the design investment and reconfiguration costs. The intelligence level investment graph, shown in figure 3.2, contain the design investment and reconfiguration cost surface as a function of intelligence and the number of configurations (expressed as R_N). This surface can be described by equation 3.4.1,

$$C_{dr}(I, R_N) = C_d + C_r R_N \tag{3.4.1}$$

where C_d is the design investment cost term and $C_r R_N$ is the reconfiguration term (equation 3.2.3). The assumption was made that the design investment cost curve of the controller, intersects the origin and increases with a polynomial function of order two, as intelligence increases. At $C_{dr}(I, 0)$, the design investment cost term is described by equation 3.4.2,

$$C_{dr}(I, 0) = aI^2 + bI \tag{3.4.2}$$

where a and b are parameters of a parabolic curve. The gradient of the design investment cost curve is also assumed to be zero at the origin (equation

3.4.3), resulting in $b = 0$. A value for a can be obtained with equation 3.4.4, by using a typical data point of C_{drTyp} at I_{Typ} In this case, $C_{drTyp}(I_{Typ}, 0) = 1000000$, at $I_{Typ} = 0.2$, was used.

$$\frac{\partial C_{dr}(0,0)}{\partial I} = 0 = b \tag{3.4.3}$$

$$a = \frac{C_{drTyp}}{I_{Typ}^2} = \frac{1000000}{0.2^2} = 25e6 \tag{3.4.4}$$

The design investment and reconfiguration cost surface are described by equation 3.4.5.

$$(I, R_N) = \frac{C_{drTyp}}{I_{Typ}^2} I^2 + \left(C_{asr} I + C_{msr} \left(\frac{C_{hyp}}{I + s} - s \right) \right) R_N \tag{3.4.5}$$

The optimal line in figure 3.2 indicates the minimum design and reconfiguration cost (9.42 cost units per manual system reconfiguration cost) and the optimal I value, at a specific R_N (e.g. 39% at 60 reconfigurations). This intelligence is used to determine the desired human effort from figure 3.1.

In summary, the hyperbolic constant (C_{hyp}) is needed to find the intelligence-effort relationship (equation 3.2.1). The autonomous and manual system reconfiguration costs are required to plot figure 3.1 (equation 3.2.2). A data point $C_{dr}(I_x, 0)$ at I_x, is required to determine the second order cost growth of intelligence. Figure 3.2 can be plotted and shows the surface minimum at a design number of reconfigurations, provide the optimal intelligence and the human effort. Intelligence and human effort are then used to guide and evaluate the design of the controller.

Chapter 4

Requirements Analysis

A requirements analysis is conducted in order to fully understand the requirements for the migration from manual to an automated reconfigurable transportation systems. A case study is first introduced on which the control, developed in this project, was implemented, and experimented with. The customer needs, and their implementation issues, are then discussed followed by the functional requirements of the target controller. Finally, technical performance measures are identified.

4.1 Case Study

The aim of the study case is to spotweld a sub-assembly of a family of circuit breakers (supplied by CBI). The sub-assembly can be seen in figure 4.1. In order to weld the parts together, they must be placed in a fixture that keeps them in place for a welding robot to weld. The fixtures are mounted on pallets that are transported on the transportation system.

A modular conveyor system was chosen from the previous work in section 1.1, before the current research, as the transportation system and is used to test the reconfigurable control software. The conveyor system can be assembled in many different configurations. Electric motors and pneumatic actuators control the motion of pallets, while sensors give feedback. The conveyor system can be seen in figure 1.1.

The pallet magazine, conveyor, feeder and welder are the other sub-systems of the manufacturing cell. These sub-systems are resources of the manufacturing cell that can complete tasks and processes on the job elements. Empty pallets (pallets with empty fixtures) are unloaded from the pallet magazine onto the conveyor. The empty pallets are transported to the feeder where they are populated with the job's parts. A populated pallet can then move to the welder and back to the feeder to be emptied. The empty pallets can then move back to be loaded into the pallet magazine. At any time, pallets can be moved to inspection resources. Note that this is a typical experimental

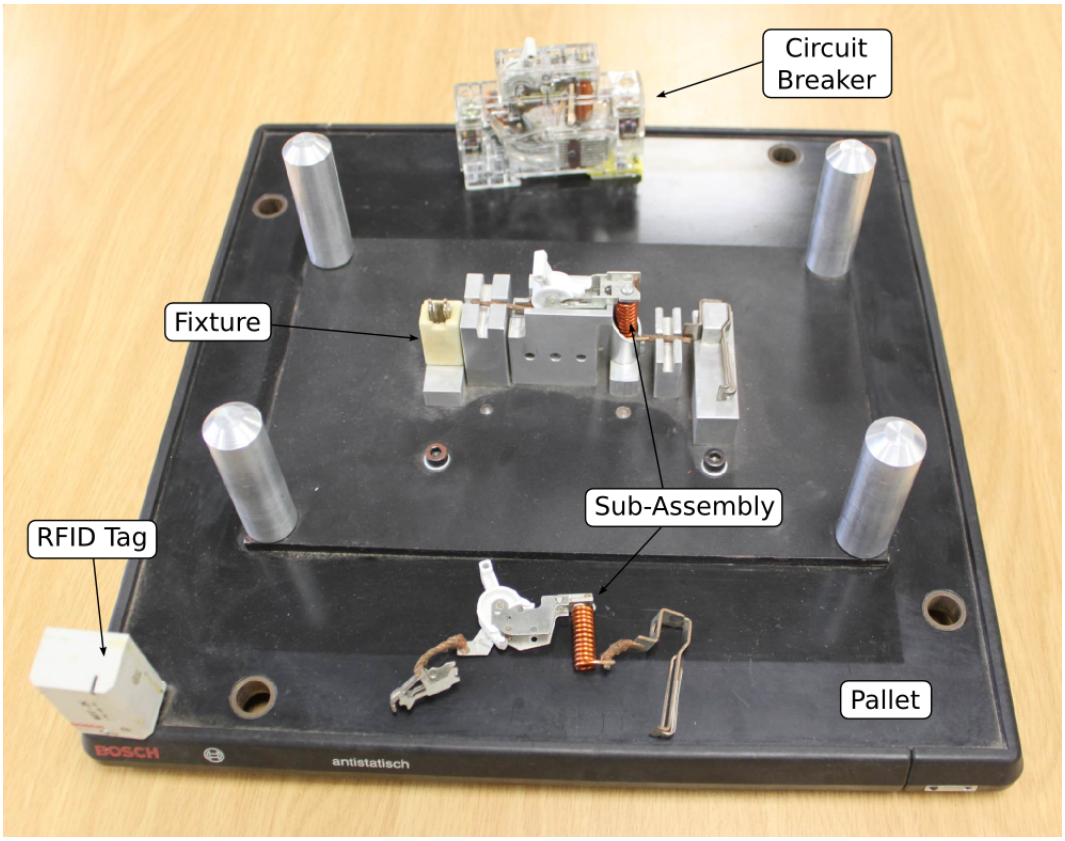
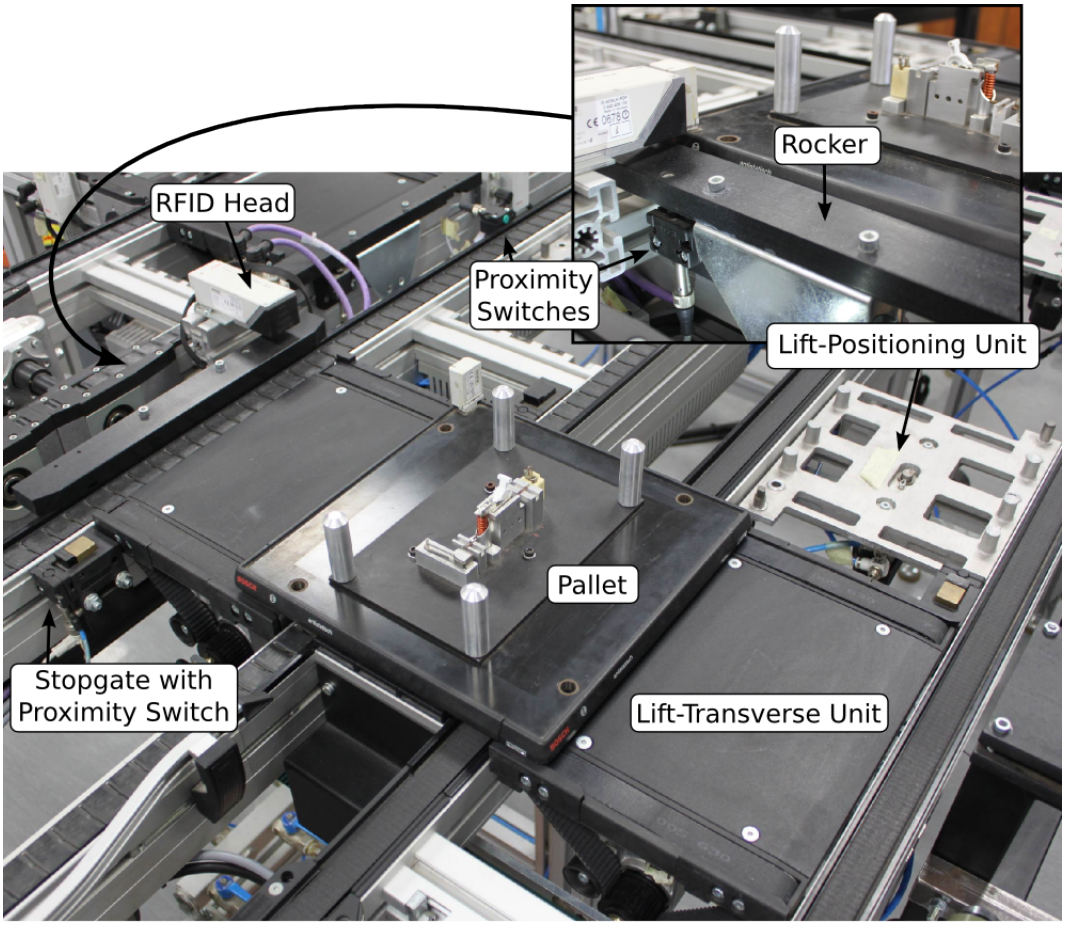


Figure 4.1: The pallet, fixture and circuitry

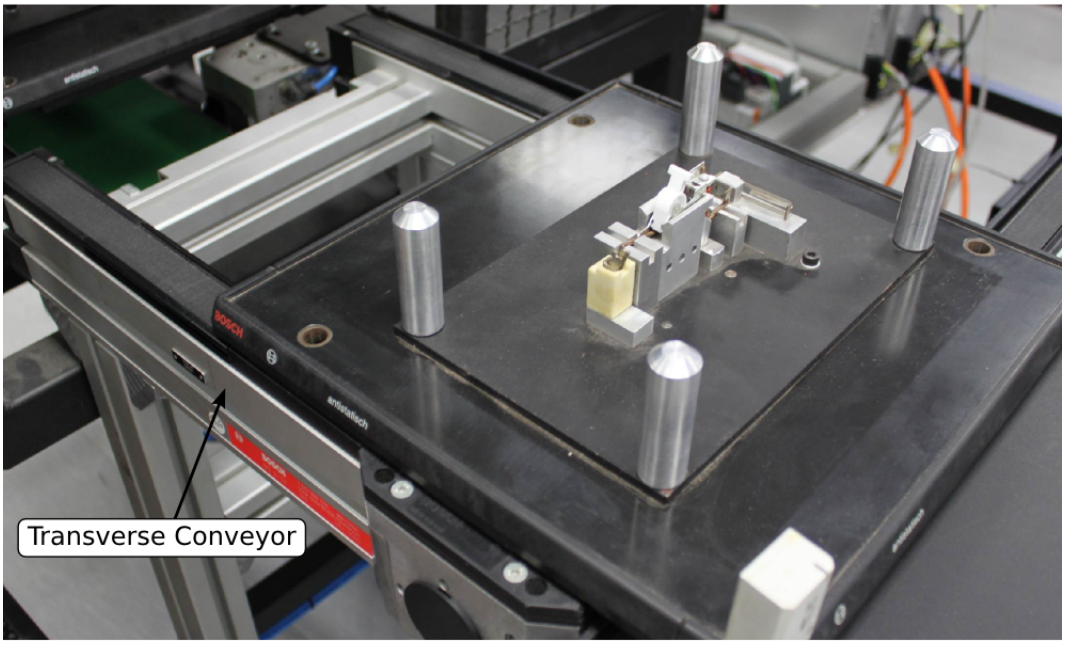
sequence, but pallets can move from any one resource to another.

The TS2plus conveyor system by Bosch Rexroth offers conveyors that can transport pallets linearly, or around bends. Lift-transverse units (see figure 4.2(a)) can lift pallets up from a conveyor and transfer it onto a transverse conveyor (figure 4.2(a) and (b)). The transverse conveyor transfers pallets between parallel conveyors and onto lift-transverse units, on other conveyors, that can lower the pallet onto the conveyor again. Lift-position units (see figure 4.2(a)) lift the pallets up from the conveyor and stabilize and orientate it for a required process accuracy. All conveyors are actuated by electric motors while the lifts are pneumatically actuated.

Stop gates are positioned to stop pallets at specific positions on the conveyor. Stop gates are pneumatically actuated and can be lowered to allow pallets to pass. Rockers are placed at the end of transverse conveyors to stop pallets in the correct positions, to be lowered by the lift-transverse units. The stop gates and the rockers can contain proximity sensors that can detect a pallet's presence. Proximity sensors can also be mounted directly on the conveyor. Additionally, Bosch offers RFID read/write heads that can be mounted on the conveyor and can read or store information on RFID tags on the pallets.



(a) Lift-transverse units transfering a pallet from one conveyor to another



(b) A transverse conveyor moving a pallet away from the pallet magazine

Figure 4.2: Hardware of the conveyor system

The pallets rest freely on the conveyor that guides it. The pallets can interface with the lift-position units, the stop gates, proximity sensors and holds the RFID tag. A steel base on the pallet offers a process area where fixtures or jigs can be mounted. Fixtures are mounted onto the steel base and contain, in this case, the circuit breaker sub-assembly.

The LLC was implemented on a Siemens S7-300 PLC. The PLC has a Siemens SITOP power supply in slot 1, a Siemens CPU315-2 CPU unit in slot 2, a Siemens DI16xDC24V input unit in slot 3, a Siemens DO16xDC24V 0.5A output unit in slot 4 and a Siemens CP343-2P AS-i input/output unit in slot 5. The electric motor drives, and other panel switches and indicators are connected to the input and output modules. Pneumatic actuators and proximity sensors were connected to the AS-i input/output module.

4.2 Customer Considerations

Manual manufacturing systems are highly adaptable (unlike automated systems) and can reach high production throughput rates, but the human presence in the manufacturing system does not allow for traceability and repeatability of products. An automated conveyor system (to serve as a transportation system in an automated manufacturing system) is to be designed to enable adaptability together with high production throughput rates. The migration from manual to automated controllers is an issue that is also addressed.

Compared to manual controllers, the development and maintenance of automated controllers are sometimes inhibited in under-developed countries (where manual manufacturing systems are still prevalent) because of the lack of:

- expertise (e.g. engineers),
- infrastructure (e.g. computer systems and networks),
- support (e.g. providers of hardware do not always have local offices in the country).

These reasons make managers hesitant to make the investment of converting to automated controllers. The automated controller must thus have a low investment risk to make the conversion from manual controllers to automated controllers attractive (see section 1.2). Designing a controller according to the HMS goals in section 2.3, will result in an automated controller with a reduced investment risk. By looking at the HMS goals, three considerations are made to lower the investment risk:

1. The HMS goals suggest the efficient use of available resources (human integration). The human brain is an excellent resource and can be used to generate a machine configuration and operational advice at low cost.

Through human integration, the manual infrastructure is not completely discarded.

2. Unfortunately a deficiency of expertise, as needed to maintain automated controller operation, still exists in under-developed countries. Few employees complete any tertiary education and controllers must be operable and maintainable with a secondary education skill level and additional training. As mentioned in section 2.4.1, safety, for real-time reconfiguration of control systems, is a concern (Brennan, 2007). With the low skill level and safety issues in mind, only off-line (non-real-time) reconfiguration is considered for the present application.

3. Proven technology, that are well supported locally, must be used to build the system.

4.3 Functional Requirements

The life cycle functional requirements are first considered. The life cycle of any system can be categorized into eight primary life cycle functions. These are development, verification, construction, deployment, training, operation, support and disposal (Department of Defense, 2001). The eight primary life cycle phases are used to define six stages that a RTS may pass through. These stages can be seen in figure 4.3.

The controller is designed in the design stage (stage 1). The development, verification and construction phase will typically be completed externally and delivered to the manufacturer. To allow the manufacturer to make the bold change to an RMS, the initial investment and the estimated return on investment must be attractive, relative to the existing manual manufacturing system. Keeping the initial cost low will result in an improved investment safety and is thus considered an important requirement. The development, verification and construction cost can be derived by the intelligence level in chapter 3.

After the design stage, the system is configured (stage 2). The ramp-up stage (stage 3), following the reconfiguration stage, is where the controller is tested for any malfunctioning, regarding execution or quality, and a fall-back to stage 2 may be necessary for reconfiguration. The completion of the ramp-up stage puts the controller into full operation (stage 4). While the controller is fully operational, it may also be converted to produce different products and product mixes. This stage may further invoke the maintenance stage (stage 5) and return to stage 4.

Deployment is only important in the pre-operation reconfiguration stage while the operation, support and training phases span the pre-operation reconfiguration, in-operation ramp-up, fully operational and maintenance stages. The existing labour force must be trained to deploy and operate the controller and must accommodate low skill levels. Reconfiguration must be simple and

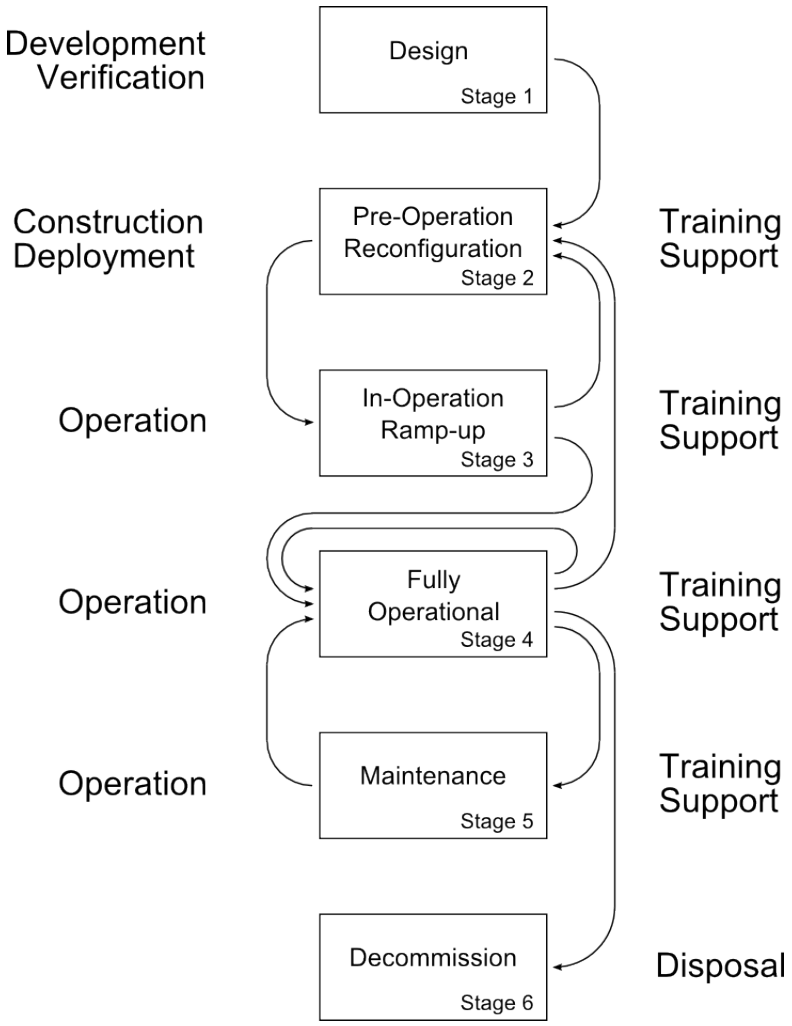


Figure 4.3: Life cycle stages

safe, allowing existing employees to be trained to complete this process in a fraction of the time needed than that of a dedicated manufacturing line redesign. Ramp-up must be simple and safe for operators to detect, diagnose and correct undesired system behaviour. When the controller is running at full operation, it must be simple and safe for the operator to convert to different products and production mixes. The monitoring of the overall system performance must also be available. The support for the controller must be available and includes maintenance, malfunction diagnostics and correction, manuals, software upgrades, add-ons, etc. Where possible, it must be simple and safe for the operator to apply the support that was received. From stage 4, the controller may return to stage 2 to reconfigure it, customizing it for a new specific production functionality and capacity. This transition (stage 4 to 2) is the essence of a reconfiguration. At the end of the system life, it is

disposed of in the decommissioning stage (stage 6).

The operational scenarios, and the requirement that they pose, are also considered. Only a few operational scenarios exist. The controller is designed to be reconfigured, powered up for production and shut down after production completion, indicating the normal operational scenario. Alternative scenarios can occur when the controller was not shut down properly (e.g. in the case of a power outage or an emergency stop). When the controller is started after an abnormal shut down, the controller state is undetermined and recovery modes must be entered for returning to the normal operational scenario. Manual control and recovery modes must thus be available to handle non-normal operational scenarios.

Another functional requirement is the need for sub-systems to communicate with each other with intra-cell synchronisation messages. This is required, for example, where a pallet must be transferred from one cell sub-system to another (e.g. from the pallet magazine, to the conveyor). To complete the transfer, the destination sub-system must prepare to receive the pallet before the cell controller sends it (e.g. the conveyor must turn its receiving conveyor on, before the pallet magazine let a pallet out).

Chapter 5

Conceptual Design

A functional analysis is initially conducted. The control architecture is then presented and introduces the two control concepts that will be tested. The reconfigurability considerations contain the bulk of the conceptual design work. Finally, a holonic architecture is introduced on which the entire controller is based.

5.1 Functional Analysis

Process flow and requirement information, gathered in the requirements analysis (chapter 4), is used to conduct a functional analysis and allocation. Figure 5.1 shows the top level functional flow of the primary functions (the life cycle phases in Department of Defense, 2001). The controller life cycle is started with development, verification, construction and deployment and end with disposal. After deployment, the controller can become operational or enter a personnel training function. The completion of operation either marks the return to operation, return to training or continue to disposal. Support is conducted simultaneously with operation and training.

The primary functions are seen in figure 5.1(a) and decomposed into the lower level functions. Figure 5.1(b) shows the second level functional flow diagram of the operation function, while figure 5.1(c) shows the third level functional flow diagram of the ramp-up and production functions. Operation includes a reconfiguration function, a ramp-up function and a production function. The ramp-up function is repeated until the controller is ready to enter production. It is shown that the ramp-up and production functions are similar. Both contain a main functional line of start-up, controller check and shut down. A manual operation, or an automatic production loop, can start after the controller check function, after which it returns to the controller check function. The automatic production loop has a conditional error correction loop appended for disturbance handling. After shutdown, the controller can be started-up again or exit the ramp-up or production function. In the case

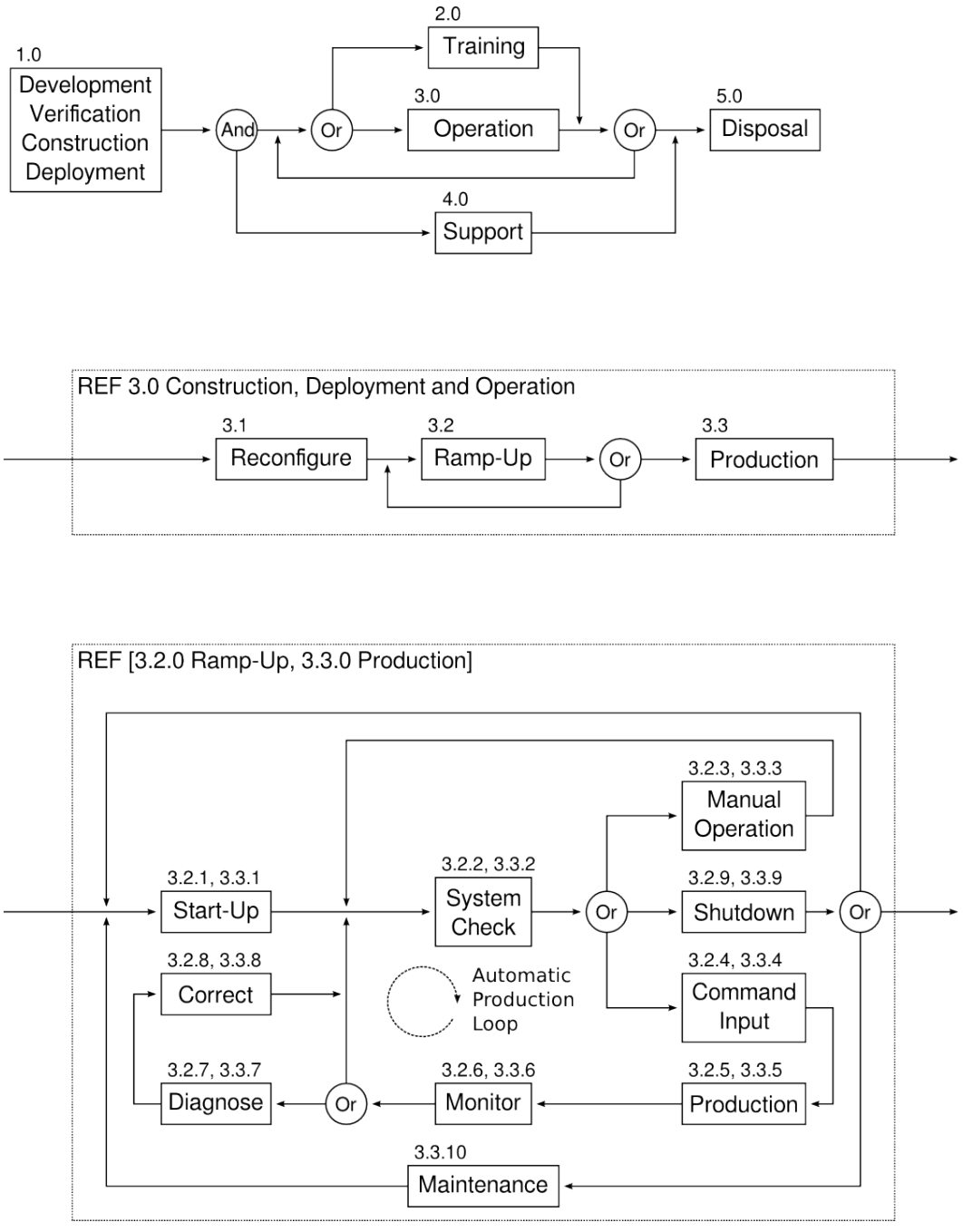


Figure 5.1: Functional analysis

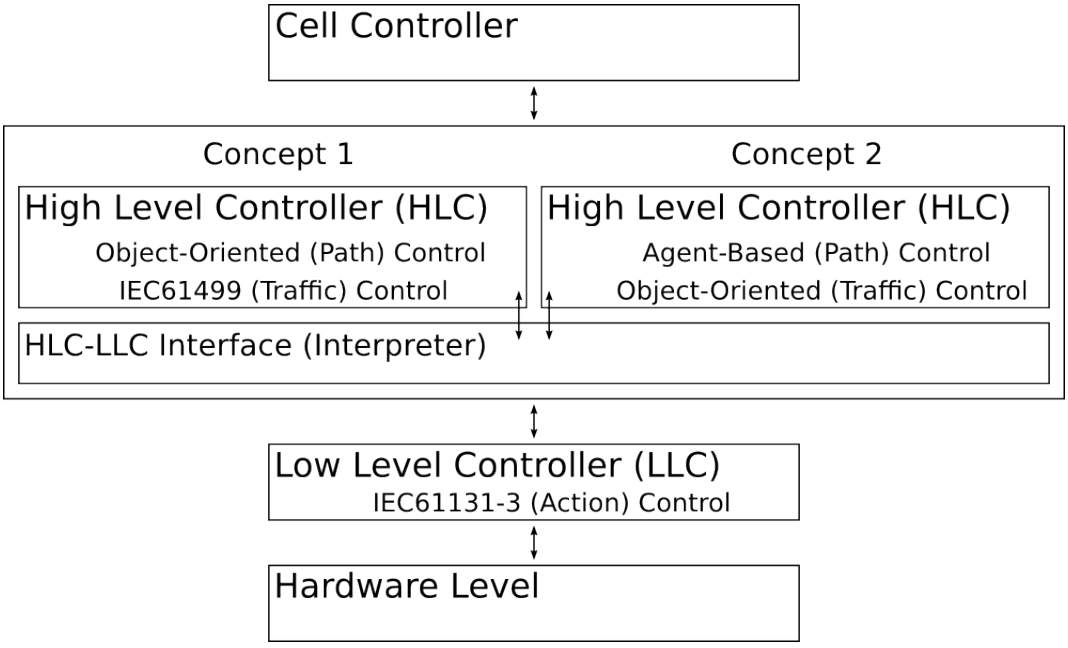


Figure 5.2: Alternative controller concepts

of the production function, the controller can be shut down and maintenance can be done before start-up.

5.2 Control Architecture

In this section, the controller structure concept varieties are introduced. The detail design of each controller is discussed in chapters 6 to 9, but a design overview is given here to set the context for the following design details of the controllers. A HLC-LLC structure is assumed, as motivated in section 2.4.1. Figure 5.2 shows the two controller concepts, considered in this thesis, and it can be seen that the cell controller communicates with the HLC, while the HLC communicates with the LLC. The LLC uses an IEC61131-3 control architecture and control the hardware. The LLC runs on a PLC and the HLCs on a personal computer. The LLC controls the hardware level.

The HLC of the first concept uses the IEC61499 function block control architecture while the HLC of the second concept uses the agent-based control architecture. Object-orientated control architectures were also implemented into the HLCs of both concepts, to add control to levels where the IEC61499 function block and agent-based control architectures are not suited to control. Note that this results in a multi-level HLC, where the top level (in the HLC) communicates with the cell controller, and the bottom level (in the HLC) communicates with the LLC.

The IEC61499 standard was designed to replace the IEC61131-3 standard and is therefore meant for low level control. However, because of hardware constraints, the IEC61499 function blocks must run on the HLC (refer to section 2.4.1). Because of the low level nature of IEC61499 function blocks, object-orientated control was added above the IEC61499 function block control to make provision for any higher level control that may be necessary. Agent-based control is considered a high level control architecture because of its high autonomous functionality and advanced communication abilities. Agent-based control is used where complex system communication and resource management, amongst others, are needed and may extend into higher levels than that of which are of concern in this study. Object-orientated control was added below the agent-based control to make provision for any lower level control that may be necessary.

For clarification purposes, design choices (controller job description), motivated in the detail design chapters (chapters 6 to 9), are presented in figure 5.2 to show what functions the different control levels fulfil. As motivated in the detail design chapters, the LLC, the HLC bottom level and the HLC top level fitted well into three major jobs that were identified. These jobs are action control (done in the LLC), traffic control (done in the HLC bottom level) and path control (done in the HLC top level). Action control takes care of triggering the required actuators in the correct sequence (chapter 6). Traffic control involves the control of traffic flow of pallets and collision avoidance, while path control does the path selection and navigation control of each pallet.

5.3 Reconfigurability Considerations

This section deals with the reconfigurability considerations in the concept design process. The intelligence level design is first discussed. This design leads to the creation of the configuration data block and, finally, an overview of the reconfiguration strategy. The concept design of the inter-level, and the intra-cell communication is also discussed.

5.3.1 Intelligence Level Design

The reconfiguration of transportation control systems always requires a measure of human input (human effort or just effort for short), the inverse of intelligence (see chapter 3). A major conceptual design concern is thus the level on which humans assist the reconfiguration. Figure 5.3 is a top view of figure 3.1, showing the intelligence-effort relationship. Before the design is started, the design starting point is considered to be point A on figure 5.3. Here the controller has no intelligence yet and needs 100% human effort in a reconfiguration. An attempt is made to increase the intelligence, moving the current design point along the intelligence-effort curve until the target design

point (point D) is reached. Point D lies on the optimized intelligence level obtained by the method in chapter 3. Note that the observation that suggests a steeper decline in effort from point A, as referred to in section 3.2. The current and target design point locations are very hard to determine. At best, an effort can be made to approach the target design point by continuously evaluating the intelligence-effort and using the results to take steps towards the goal.

The required human effort in the reconfiguration of controllers can be reduced by isolating controller configuration data from the control program structure (see section 6.1 for motivation). The control program becomes a module with the configuration data block acting as its interface with human input. The human effort in reconfiguration is now only concerned with the updating of the configuration data block, and can be kept simple for minimum human effort. The configuration data block is designed to be a human readable file, allowing the controller to be a closed module with which the operator can interface with, from a variety of environments, improving modularity and integrability. The design steps to approach the intelligence-effort design point are:

1. The first design step (from point A to B in figure 5.3) is thus to separate system configuration data from the control program structure. This step means that the system configuration is imported into data structures and scanned by the control program to make decisions and act upon them. This approach is an alternative to hard-coding the configuration data into the control program. A small increase of intelligence is needed (the need for indirect addressing), but the external configuration of a closed control program can be done, resulting in a major reduction in human effort. This change correlates with the intelligence-effort curve in figure 5.3, and can be indicated by point B.

2. As discussed in section 2.4.1, controllers are designed to be composed of multiple levels. Thus, a further step is made towards point C, by creating a general data format (see section 5.3.2) from which the configuration data block, of all control levels, can be build. This requires a small increase of intelligence (the need to building data structures from a data table), but allows the modification of all control levels from a single location, resulting in a reduction in human effort. The ability to control all levels from a single location enables the next step.

3. A final step can be made towards point D by adapting the configuration data block to be processed in a computer aided configuration environment. The user can reconfigure the system in a graphical environment which can automatically generate input files that are downloaded onto the controllers. But to achieve such versatility, the controller must be built-up using intelligent building blocks. This results in a significant

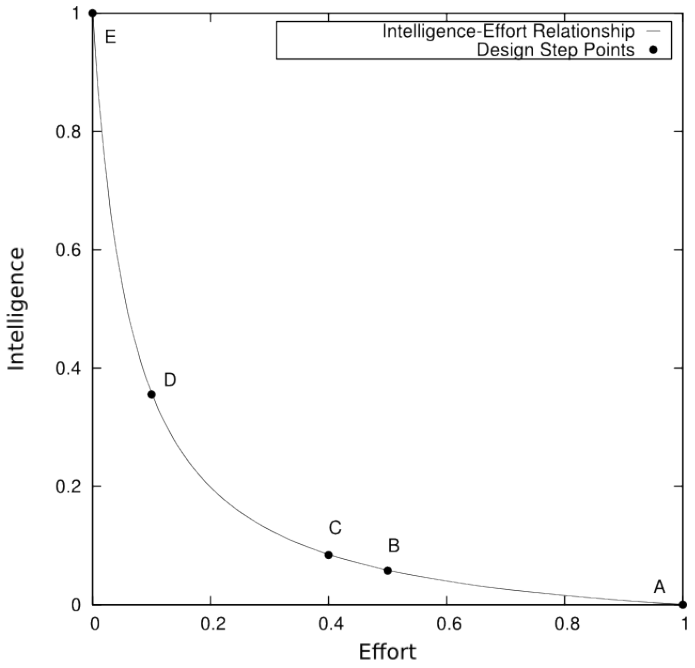


Figure 5.3: The intelligent-effort relationship graph

step in intelligence (object-oriented programs, the need for a GUI, data delivery management), but allow for the reconfiguration of all control levels in a simple environment, resulting in a comparable decrease in human effort.

5.3.2 Configuration Data Block

A major challenge is to create a general data format that contains all the information necessary to configure all control levels with a single control interface, yet be compact enough for human understanding and modification. An abstraction is defined by the Collins English Dictionary as:

> "the process of formulating generalized ideas or concepts by extracting common qualities from specific examples"

and

> "an idea or concept formulated in this way".

Transportation systems have common objectives and a generalized concept (abstraction) can formulated. The controller is developed in a way to control an abstraction of the transportation system, rather than specific transportation system hardware. The transportation abstraction presents only a model

of the hardware to the controller, and can reduce the control work and possibly make the controller hardware independent. All transportation systems have the common objective of transferring (transitions) materials, parts or products (tokens) between process locations (positions). An abstraction of transportation is therefore formulated using a Petri-Net model (see section 2.4.1), on which the Petri-Net has positions that represent loading areas, buffer areas, process stations and intermediate transition steps. The positions are connected with transitions that represent the actions necessary to move a token from one position to another, when the transition is fired (triggered).

The idea of intermediate transitions is one of the major general data format enablers offered by the transportation abstraction. The transportation process, from one resource to another, is often a complex sequence of time dependant actions. Such sequences are decomposed into a series of discrete transition steps, where single time dependant actions are individually considered and combined, with others, to build complex sequences. Another general data format enabler is the ability to decompose and classify complex systems into a few groups of common entities. The abstraction allowed the description of the entire conveyor system, with various components, in three tables that are shown in section A.2. The tables are the:

- the transition table containing the information needed to describe the Petri-Net structure,
- the position table containing detail information of the positions in the Petri-Net and
- the action table contains detailed information regarding the connectivity of actuators and sensors.

5.3.3 Reconfiguration Strategy

Figure 5.4 shows the computer aided configuration environment containing a modelling user interface, a configuration user interface and the compiler. The operator can build the system model in the modelling user interface. The configuration user interface can be used to change the configuration properties, like starting addresses of data in the memory. The compiler uses inputs from the user interfaces to compile the configuration data block (introduced in section A.2). A control user interface in the HLC reads the configuration data block and writes it to the configuration processor.

The configuration processor is the object that reads the configuration data block file and builds the system map in the HLC. The configuration processor understands the general data format and reads the configuration data block file immediately after it is instantiated. The system map is built as a structure of objects that are instantiated as the configuration data block file is read.

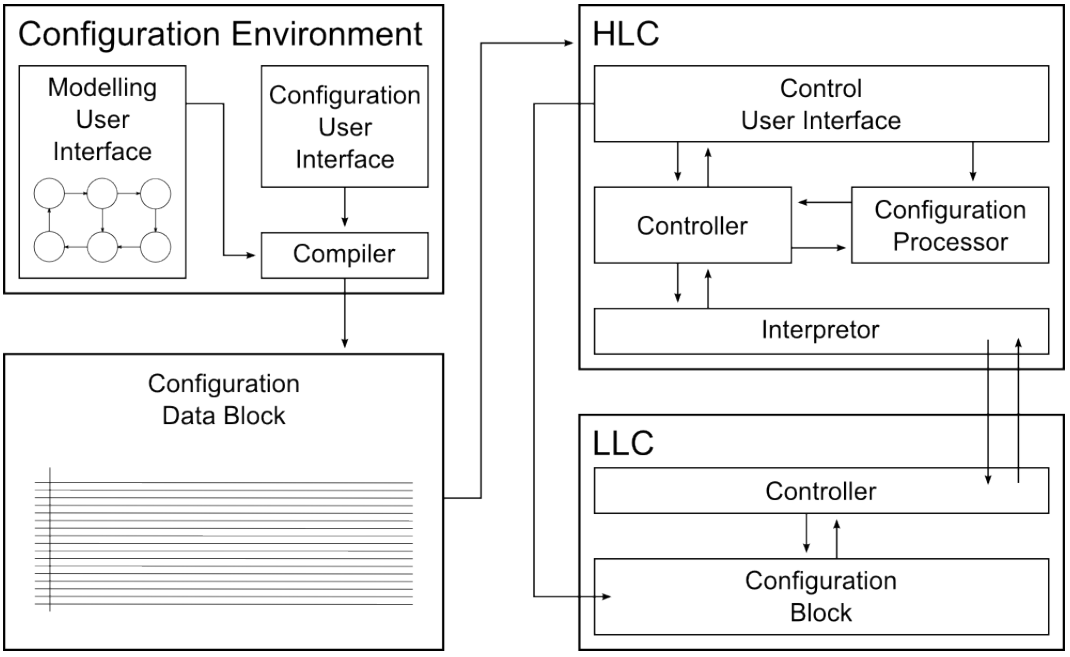


Figure 5.4: The reconfiguration strategy

The objects (positions and transitions) are linked according to the Petri-Net modelling scheme.

The HLC user interface loads the LLC configuration data block into an area in the LLC memory, designated the configuration block. The HLC controller is operated from the HLC user interface and exchanges data with the configuration processor to control the LLC. The LLC communicates with the HLC via the interpreter (discussed in section 5.3.4) The LLC exchanges data with its configuration block and controls the hardware. The design decisions that are made in this chapter, are consistent with the code generation/download paradigm that Vrba *et al.* (2011) describe (refer to the external modelling method discussed in section 2.4.1).

Section 2.4.1 describes the LLC responsibilities. The following LLC objectives are obtained from the responsibilities that:

1. The LLC must ensure a degree of protection against dangerous actions from the HLC.

2. The LLC interaction with actuators and sensors on the hardware level and must be reliable and real-time response.

3. The Device failures must be diagnosed and reported by the LLC.

4. The LLC must provide efficient interaction with humans and other parties.

An external signal (from the HLC or other party) modifies a variable in the LLC memory and the LLC will start the corresponding sequence of actuations as soon as the variable change is encountered. The LLC must be designed to always operate in a safe way with any external signals. Conveyor systems are inherently safe since pallets will just stop against each other if the wrong action is issued. With AGV or robotic systems, the LLC controlling these systems must be able to avoid collisions that may result in major damage. Chapter 6 describes the detail design of the LLC.

The HLC is seen by Vrba *et al.* (2011) as an intelligent optimization layer sitting on top of the LLC. While the LLC is operational, providing minimal functionality, the HLC can connect to the LLC and control it. A HLC connection failure will leave the LLC in a safe state. In this way the HLC is decoupled from the LLC, and can freely connect and disconnect without resulting in major failures or any damages. The HLC focuses only on the transition and position tables and is configured to read and write to the transition, position and status variables used in the main program of the LLC.

5.3.4 Inter-Level Communication

A challenge that accompanies multi-level control is the requirements for additional inter-level communication. Some communication options are Profibus, Ethernet and RS232. Ethernet communication is not hard real-time, like Profibus and RS232, but is easily reconfigurable since multiple devices can simply be connected to an ethernet switch device. The optimal bus selection is not the focus of this work and since all control devices (computers and PLCs) are Ethernet ready, an Ethernet network was used between all computers and PLCs.

Since a PLC was chosen for the LLC (see section 4.1), a reconfigurable command interpretation solution is required for communication between the computer-based HLC and the PLC. PLCs work on a "bit" level and commands are usually sent by setting boolean values (or bits). The PLC can be considered to work with bit-based commands because most control on PLCs work by setting and clearing bits. The use of values (on PLCs) are less common, and more so, the use of strings. Reading and writing bits on the PLC requires string sized commands in the order of 10 characters (10 byte or 80 bits). Commands of up to 265 characters are not uncommon to enable easy human interpretation of the same commands. Unlike PLCs, computers are adapted to support such commands, defined as string-based commands.

Strings can be interpreted into bit commands, and vice versa, by assigning string commands to bits. The OPC Foundation (www.opcfoundation.org) has created a standard to which PLC manufacturers can conform to. An OPC server is used by running it on a computer and connecting to conformed PLCs. The solution that OPC servers have to the command interpretation problem is to define "tags" to memory addresses (bits or bytes). The tags are defined in a

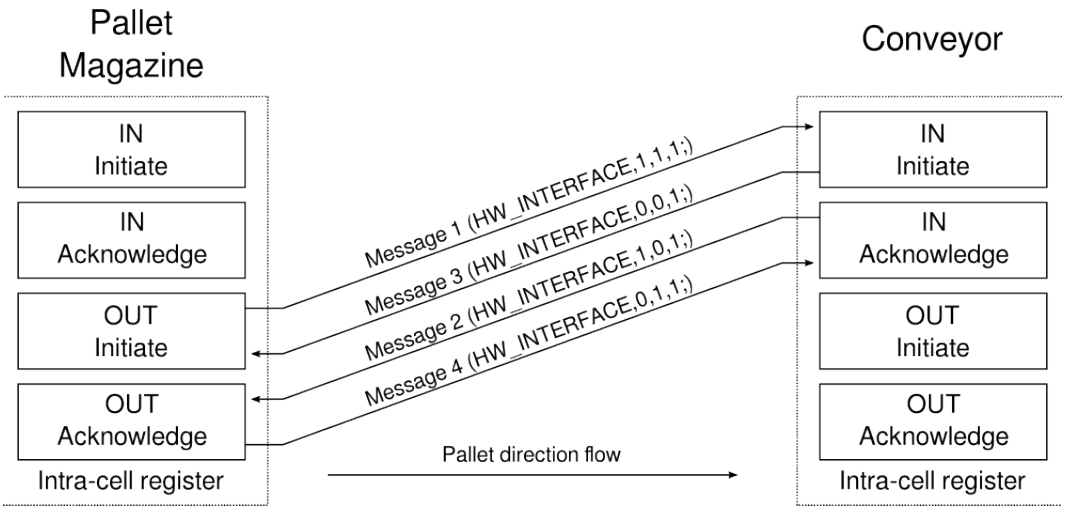


Figure 5.5: Hardware interface communication

configuration environment of the OPC server. Although a library (LibNoDave) was used rather than OPC (as motivated in section 7.3.3), the solution in this study functions on the same principal.

The interpreter is defined as the program that reads commands from the HLC, interpret the commands (convert string-based to bit-based) and deliver the commands to the LLC. The interpreter can also detect changed bits on the LLC, interpret the bits to string-based commands and deliver the commands to the HLC. The HLCs are designed with fixed string commands that are assigned to addresses on the PLC, by the interpreter. The interpreter can assign the string commands to the PLC addresses by reading the interpretation from a descriptor file. The descriptor file is also generated from the same configuration environment used to compile the configuration data block. The descriptor file contains lines that each allocates strings (descriptors and parameters), to PLC addresses, as well as the actions to be conducted at these addresses. The descriptor file format is defined in section A.4.

5.3.5 Intra-Cell Synchronization

It is seen in section 5.4 that operational holons can communicate with each other. This communication enables sub-systems to communicate with each other and execute a hardware interface procedure. The hardware interfacing procedure is a sequence of messages that are passed between two sub-systems and allow the pallet, in this case, to pass from one sub-system to another. In the case where the pallet magazine is commanded to pass a pallet to the conveyor system, a message must be sent to the conveyor, telling the conveyor to prepare for receiving the pallet. The conveyor must switch its receiving conveyor on and send a ready message to the pallet magazine. The pallet

magazine will unload a pallet onto the conveyor and the pallet will be conveyed to the receiving position on the conveyor. As soon as the conveyor senses the pallet in the receiving position, the conveyor sends a "pallet-received" message to the pallet magazine, after which the pallet magazine replies with a "done" message. The message sequence can be seen in figure 5.5 and is discussed later in this section.

A decision had to be made on what communication path to use for the sub-system communication. Two options are that:

1. each sub-system must have a physical connection (using the PLC's input/output modules) with other sub-systems and communicate directly with each other or

2. communication can be routed through the existing communication network used to communicate with the supervisor holon.

Each method requires the reconfiguration operator to supply sub-system connection information, either to the sub-system, or to the cell controller. Direct physical connections require the sub-system connection map to reside with the sub-system. This makes the sub-systems more autonomous since each one holds its own view of its surroundings, not being blind as with the case where all connection information resides with the cell controller. The added direct physical connections do, however, increase the controller reconfiguration cost. The operator must connect additional communication lines between the sub-systems and each controller must manage the communication lines. For these reasons, the use of the existing communication network was chosen, as the preferred sub-system communication method.

Each sub-system has multiple intra-cell registers (ten in this case) that are written to and monitored by the interpreter, which generates messages to be sent to neighbouring sub-systems. Each intra-cell register has four bits (boolean variables), which are two inbound and two outbound bits. The two inbound bits are used when a pallet is coming into the sub-system, while the two outbound bits will be used to send a pallet out of the sub-system. The inbound and outbound bits do each have initiate and acknowledge bits. The interpreter, from section 5.3.4, is also set-up to monitor the four bits. As soon as any of the bits change, the interpreter generates a message (a string) that is send through the HLC to the cell controller. When the cell controller recognizes the message, it looks at an internal sub-system connection map, and sends it to the correct destination sub-system.

Figure 5.5 shows the messages that modify the intra-cell port variables in the process of sending a pallet from the pallet magazine to the conveyor. In order to do this, the pallet magazine controller sets its `OUTinit` variable. The interpreter of the pallet magazine detects the variable change and sends message 1 to the cell controller. The cell controller uses its sub-system connection map (table 5.1) and redirects the message 1 to the conveyor.

Table 5.1: Sub-system connection map table

Internal hardware register numbers					
	1	2	3	4	5
Sub-system numbers					
1 (Pallet Magazine)	2				
2 (Conveyor)	1	3	5	5	4
3 (Feeder)		2			
4 (Welder)					2
5 (Inspection)			2	2	

On receiving message 1, the `ININit` variable of the conveyor is set, and it prepares to receive a pallet by starting its conveyor. The `INackw` variable of the conveyor is then set and the generated message 2 sets the `OUTackw` variable on the pallet magazine. The pallet magazine controller uses the rising edge of the `OUTackw` variable to start its output conveyor that will move the pallet out of the pallet magazine. When a sensor on the conveyor sub-system indicates the presence of the pallet, its `ININit` variable is cleared and message 3 clears the pallet magazine's `OUTinit` variable. This indicates the stopping of the pallet magazine's output conveyor and clears its `OUTackw` variable which sends message 4 to clear `INackw` on the conveyor sub-system. The format of the hardware interface messages is `HW_INTERFACE,setBit,dirBit,port;`. The `setBit` field indicates whether the destination variable must be set (if 1) or cleared (if 0). The `dirBit` field indicates the direction of the message. A `dirBit` of 1 indicates a request message (a message in the same direction as the pallet). A `dirBit` of 0 indicates a reply message (a message back from the pallet destination sub-system). The `port` field indicates the port number.

Table 5.1 shows the sub-system connection map. Each sub-system has entries indicating the numbers of the sub-system(s) to which it is connected. These sub-system numbers are linked to the port numbers which are entered in the `port` fields. When the cell controller receives a hardware interface message, the cell controller looks at the entry at the row of the sub-system (of which the message came from) and the column indicated by `port`, and forward the message to that sub-system.

5.4 The Holonic Architecture

As discussed in chapter 4, the RTS is developed using the holonic architecture concept of HMSs. The reconfigurable manufacturing cell, of which the RTS is a sub-system, has a cell controller in the top level holarchy (illustrated in figure 5.6) that controls all the sub-systems in the cell. The manufacturing cell contains the conveyor, pallet magazine, feeder and welder sub-systems.

The ADACOR holonic reference architecture is chosen above PROSA (see

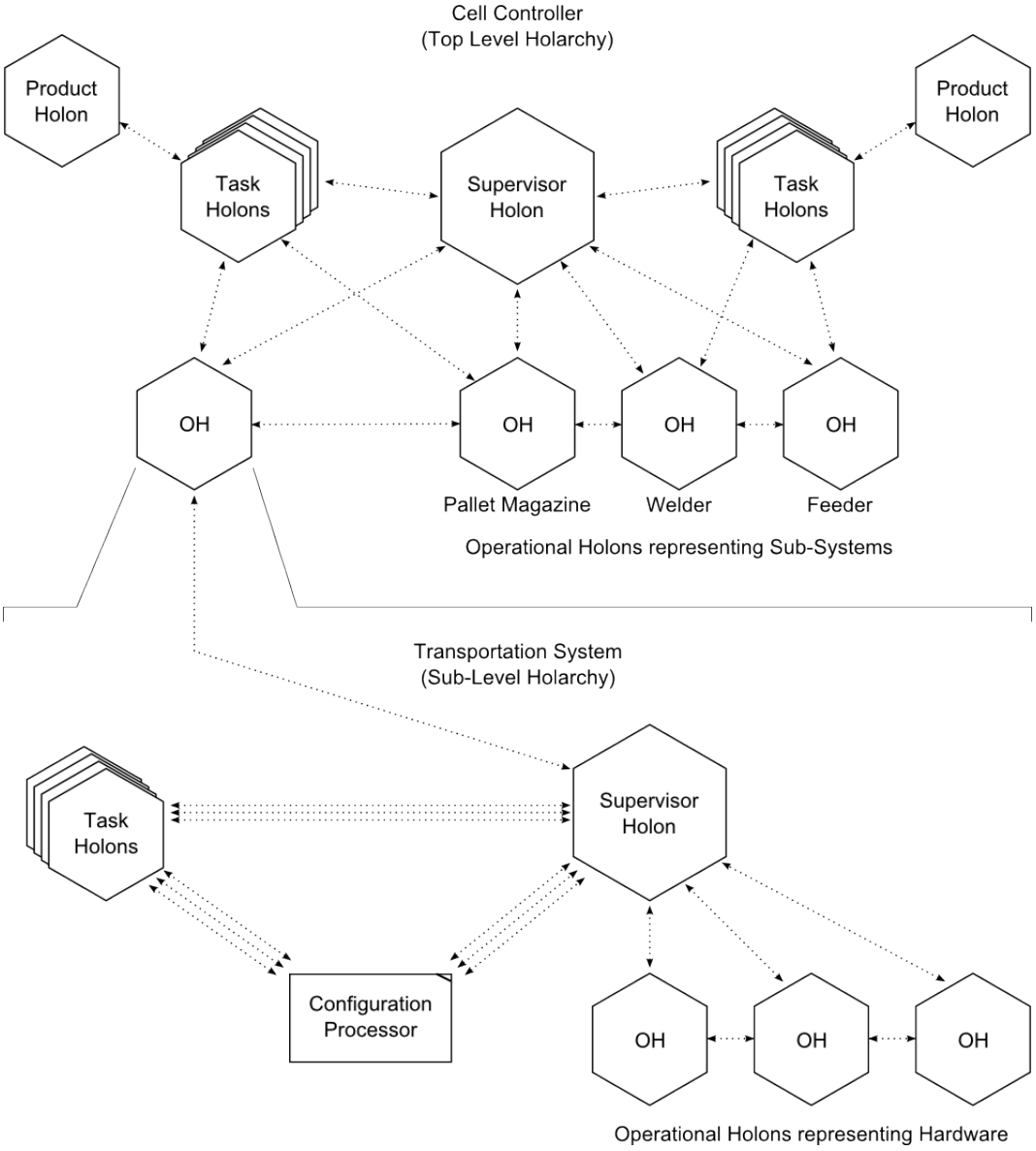


Figure 5.6: The ADACOR holonic architecture

section 2.4.4) because of the supervisor holon that offers global optimisation. One of the cell controller's operational holons, which corresponds to the transportation system, is implemented as a sub-level holarchy. This sub-level holarchy contains its own supervisor, task and operational holons. Product holons are not applicable in this transportation system since there is no product information necessary for transportation systems where the content of the pallet does not influence the transportation path or sequence. Task holons, on the other hand, are necessary since the nature of transportation is a process of tasks that must be scheduled and rescheduled when they fail.

The ADACOR architecture, implemented as in the sub-level holarchy, is illustrated in the bottom of figure 5.6. The sub-level holarchy supervisor holon communicates with the top level holarchy and the operational holons in its own holarchy. In both concepts the operational holons are represented by the configuration processor object (in the top HLC level of concept 1 and in the bottom HLC level of concept 2). The configuration processor object can be seen as a mirror that keeps a copy of the operational holon data (e.g. their state information). Task holons only communicate with the operational holons in the sense that they can communicate with a reflection (synchronized data between the PLC and configuration processor) of the operational holons, on the configuration processor, and advise the supervisor in commanding them.

The HLC communication module (the interpreter introduced in section 5.3.4) in the supervisor holon monitors changes in LLC. These changes are then reported to the configuration processor to update the variables in the objects. The supervisor and task holons can invoke the configuration processor to enquire any information. Task holons can, for example, ask the configuration processor which transition can move a token, from the current position, to the next, on a path between two resources. The path table is contained and managed in the configuration processor. The supervisor creates and destroys task holons, as well as command task holons to attempt transportation jobs, putting them in contact with the elements of the configuration processor.

It is important to note that the communication between the task holons and the operational holons, through the supervisor holon, is a deviation from the ADACOR architecture. To explain the reason for this deviation, the origin of the configuration processor must be considered. An obvious container for the configuration processor is the supervisor holon, since it will initiate the other holons and stay active, while connecting to other holons thought the control session. Because the supervisor holon contains the configuration processor, inter-holon communication could be used between the task holons and the supervisor holon.

The supervisor holon writes directly to the operational holons in the configuration processor, allowing task holon messages to be sent to the operational holons, within the same controller. The alternative method is to allow each task holon to open a socket to the lower control level. The lower control level will have to manage multiple connections from the task holons and is a

complexity that requires, amongst others, more advanced client-server socket management. This option may be suggested as further research. Operational elements also communicate with each other to ensure that their actions are allowed by neighbouring holons.

Because of the supervisor centred architecture of sub-level holarchies, all control events are assumed to be initiated by the supervisor holon. This assumption had to be made since, according to the author's judgement, the true ADACOR architecture requires a higher level of computer science knowledge, than that which falls in the scope of this work. This assumption simplifies the controller, but is undesirable for RMS and HMS, because of the supervisor centred structure. Further work is proposed where the task holons use the supervisor to control the operational holons only initially, and then take the control over from the supervisor (as is closer to the ADACOR architecture understood by the author).

Chapter 6

IEC 61131-3 Low Level Controller Design

6.1 Design Strategy

The LLC is illustrated in figure 6.1. The first objective of the LLC is action control which includes the:

- activation of an actuator, or a sequence of actuators, from an incoming command,
- reporting of sensor information to an external controller and
- basic control of actuators with the LLC sensor, status and command information.

The software of classical PLCs is standardized with IEC61131-3, which has properties like global variables and data-driven control, resulting in low reconfigurability (refer to section 2.4.3). Reconfigurable IEC61131-3 controllers can be designed by considering the PLC controller as a closed module, connected to an external configuration environment (as explained in section 5.3.3). The configuration environment can compile and download the configuration data into an accessible memory area on the PLC. After implementing the objective of action control on the PLC, the complexity of the control code started nearing the functionality limit of the PLC. The design led to an approximately 80% of the random access memory usage and resulted in a structured text language code size that became difficult to manage. Although there might be advancements that can be made to use less memory or better manage the code, a decision was made to allocate the next objective, traffic control, to the next level of control (the bottom level of the HLC).

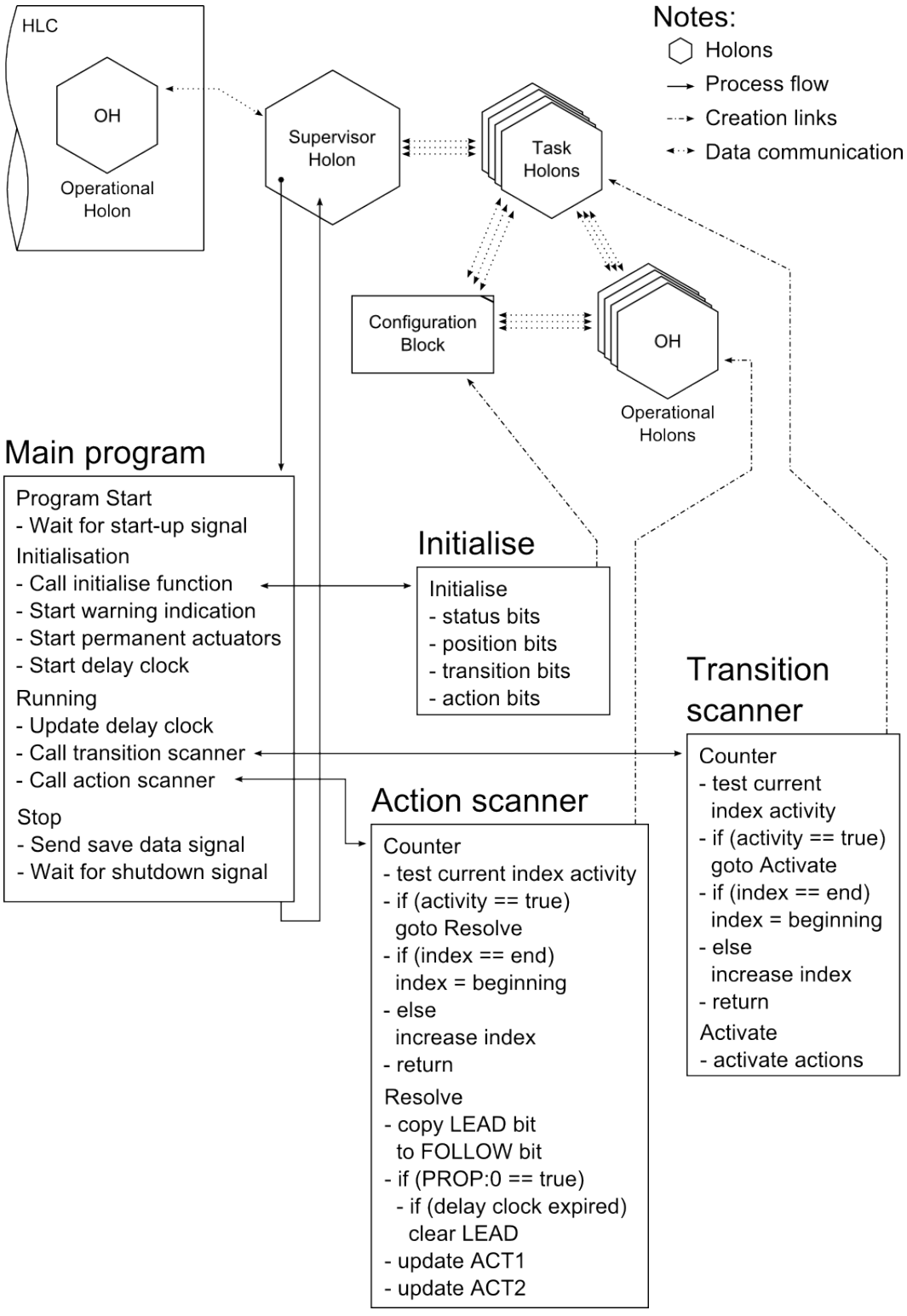


Figure 6.1: LLC program layout

6.2 The Object-Oriented Design Modelling and ADACOR Architecture

The object-orientated design methodology described by Vrba *et al.* (2011) (section 2.4.3), proposed the following enhancements that was used to create holon templates. Indirect addressing allows linking and usage of dynamic, moveable and changeable data. If a controller can be linked to data that is flexible, reconfigurable systems can be represented by such data. Another technique is the containment (or combination) of multiple elementary actions to produce more complex actions. Macro instructions are used by copying referenced data into its working memory field and executing the macro. The holon templates are then used to construct the ADACOR architecture (section 5.4).

The configuration data, described in section 5.3.2, are downloaded into the configuration block of the PLC. The main program initialise the status, position, transition and action bits, which are used to index through the configuration block. Operational holons are represented by the action entities (the row entries of tables A.3 and A.4). These operational holons can each control an actuator, a sensor or an internal variable. Operational holons can also communicate with each other. The transportation entities (the row entries of table A.1) are represented by the task holons. Task holons are linked to single or multiple operational holons and can initiate actions.

6.3 Implementation

Once the user or the HLC sends a start signal, the main program continuously scans through its networks, jumping to the different modules, including the data scanners. The interpreter (in section 5.3.4) keeps the LLC state updated in its memory. The main program is built using ladder logic and function blocks, while the data scanners are written in statement list language. The reason for this design decision is that the ladder logic and function block alternatives do not support the use of indirect addressing.

The human interface to PLCs is usually toggle switches, buttons and/or indicator lights, connected to its input/output units. The LLC has a toggle switch for a power switch, two buttons to set the LLC to the running or stopped modes and an indicator light that indicates running mode. These controls are all mounted on the front of the control panel that contains the PLC and other electrical devises.

The first data scanner (the transition scanner) indexes through the table of transitions and triggers any transition that is activated. The table with transition information, is downloaded onto the LLC, from table A.1. The table contains:

- the address of a bit indicating the presence of a pallet in the start position (`SPOS.PRES`),
- the trigger address of the end position (`EPOS.ADDR`),
- the trigger address of the transition (`ADDR`),
- and the activation addresses of the actions (`ACT.LEAD` and `!ACT.LEAD`). Note that the `LEAD` bits of `ACT` and `!ACT` are set and cleared, respectively.

The transitions are triggered when, either, both `SPOS.PRES` and `EPOS.ADDR` are true, or `ADDR` is true. When a transition is triggered, an activate method is called to activate actions.

The second data scanner (the action scanner) indexes through the table of actions and resolves any action that is activated (unresolved). The table with action information, is downloaded on the LLC, from tables A.3 and A.4. The table contains:

- the address of a leading bit (`LEAD`),
- the address of a following bit (`FOLLOW`),
- the addresses of two action bits that are to be triggered (`ACT1` and `ACT2`) and
- property bits (`PROP`).

When an action is activated, it becomes unresolved because of the leading bit that is not equal to its following bit. In such a case, the following bit will be set equal to the leading bit and the action bits will be set, cleared or remain unchanged, depending on the property bits. The action can also be unresolved if the leading bit is set, the first bit in property bits is set and a delay clock has completed one cycle. This will cause the leading bit to be cleared, making the action unresolved yet again, to be handled in the next scan. Note that the delay clock is a bit that is continuously set and cleared for specific times. The delay clock runs in the main program and its set and clear times are adjusted to suit the hardware application.

Chapter 7

IEC 64199 Function Block Controller Design

Vyatkin (2007) discussed the IEC61499 standard and uses the software tool Function Block Development Kit (FBDK), to implement it. He suggests the use of the MVC engineering methodology, as a method by which a function block controller can be designed. Because only a few PLC devices support the new IEC standard, a computer is used for this controller. Thus the IEC61499 function block controller is built directly above the PLC-based LLC, in the lowest level of the HLC (the bottom HLC).

7.1 Design Strategy

The IEC61499 function block controller (referred to as the function block controller) can be seen in figure 7.1, and is the bottom HLC of concept 1 (figure 5.2). As described in section 6.1, the objective of the HLC begins at the traffic control of pallets, ensuring that no collisions occur. This is done by considering the transitions to be fired, keeping track of the number of tokens in each position in the Petri-Net and ensuring that the start position has pallets to send and the end position, open space to receive them. Transitions must also be denied if a position is blocked by another position.

After implementing the traffic control with function blocks, the complexity of the function block networks became excessively large and a decision was made to implement the path control objective in the in the top HLC level. The controller is developed by considering the strengths of IEC61499 described in this section. Hardware simulation and modelling is one strength of IEC61499 but is not implemented since simulation does not fall in the scope of this study (see section 1.3).

Programmability is one advantage of the IEC61499 function block standard. One option is to design the controller by mainly building it up from the simple function blocks provided in FBDK. An alternative is to create custom

function blocks that contain all the code necessary to do the same job. Creating fewer custom function blocks, which contain more complex code, makes the function block network smaller, but eliminates the possibility for an operator to modify the internal code without major programming experience. Using the standard set of provided function blocks may enable the user to enter the actual control code and make modifications, with little training. Therefore, the controller was designed using a standard set of provided function blocks, allowing the exploration of the programmability functionality.

Encapsulation, portability, interoperability and configurability (section 2.4.5) are used in the following ways. The hardware level (the LLC in this case), as well as the communication with other control units, are encapsulated with communication devices (section 7.3.2). Composite function blocks are further used to encapsulate networks of function blocks provided by FBDK. The server and client function blocks provided by FBDK were used, instead of creating custom service interface function blocks.

A system file (an *.sys file) describes the function block network system (where the system here indicates the entire function block program) in an XML format. The system file is an editable text file, and can be directly compiled from the computer aided configuration environment in section 5.3.3.

7.2 MVC Engineering Methodology

The function block controller design is discussed in the light of the MVC engineering methodology (summarized in section 2.4.4). The example applications that IEC61499 is used for (by Vyatkin (2007)) are systems like milling machines, that typically have a number of actuators and sensors in the order of 10. The examples are also hardware specific, having function blocks for drills, slides, conveyors, etc. For the work presented here, building views with elementary HMIs, animating them and creating models and model HMIs of the actual hardware configuration, poses a major challenge since many more actuators and sensors are considered.

The design for a generic controller requires the specific view and model design of machine components (function blocks). Such design needs a high level of expertise, resulting in high reconfiguration costs. Generic views and models that represent transportation systems (like Petri-Nets) may eliminate this cost. Additionally, systems with actuator and sensor amounts, in the order of 100, requires automated view and model building in the case of a reconfiguration.

The three layers (HMI, control and interface layers, from top to bottom in figure 7.1) make up the IEC61499 controller. The MVC methodology was used to first design the HMI, followed by the design of the control layer, and finally the interface layer. Generic views and models deals with hardware simulation, and does not fall into the scope of this study.

7.3 Implementation

The controller consists of a HMI device (HMI), the control device (HL_Control) together with the interface device (HLXInterface) for communication with the top HLC and the hardware interface device (LLInterface). Figure 7.1 presents each device in the three MVC layers with their contained resources and accompanying GUI frames.

The process flow of IEC61499 function blocks is discussed next. IEC61499 uses an event-based process flow mechanism that follows only specific process flow routes. The process flow of the controller is thus protected by the limited effect that changes, by an operator, can have on the controller process flow. For this reason, it is safer for an operator to modify IEC61499 function blocks than IEC61131-3. HLC software is implemented on a standard personal computer. All HLCs were tested on a Toshiba Satellite S300 laptop with an Intel Core2 Duo CPU T6500 2.10GHz 2 processor and 4Gb of memory. The operating system used was Ubuntu 12.04 LTS 32-bit. The IEC61499 function block HLC was implemented in Java with FBDK. FBDK offers a graphical interface for creating function block networks as well as the function blocks themselves and their components.

7.3.1 HMI Device

The HMI device contains two panel resources (ModeC and StatusP) and one embedded resource (GetI). The ModeC network allows the user to set the IP addresses of the LLC and external HLC interfaces, as well as the operating mode of the top HLC (see section B.1.1). The StatusP resource enables basic access to the LLC. The LLC can be started and stopped, and an indicator provides limited diagnostics by displaying the system running state.

The GetI resource subscribes to the information packets from the LLC, processes them and publishes the information for diagnostics in the StatusP resource. The HMI is built using the standard set of function blocks and can be modified and maintained with little training and support.

7.3.2 Control and Interface Devices

The control device contains a panel resource and further embedded resources. The panel resource (Panel) is a GUI that represents the controller device. Test buttons are created on the panel resource and allow the operator to generate test transition signals and set initial values. These buttons can be added manually by an operator, since it is created by standard function blocks that are added and connected to the network. The functionality of adding and removing buttons, demonstrates the ability to easily modify a controller and add test inputs to generate test signals.

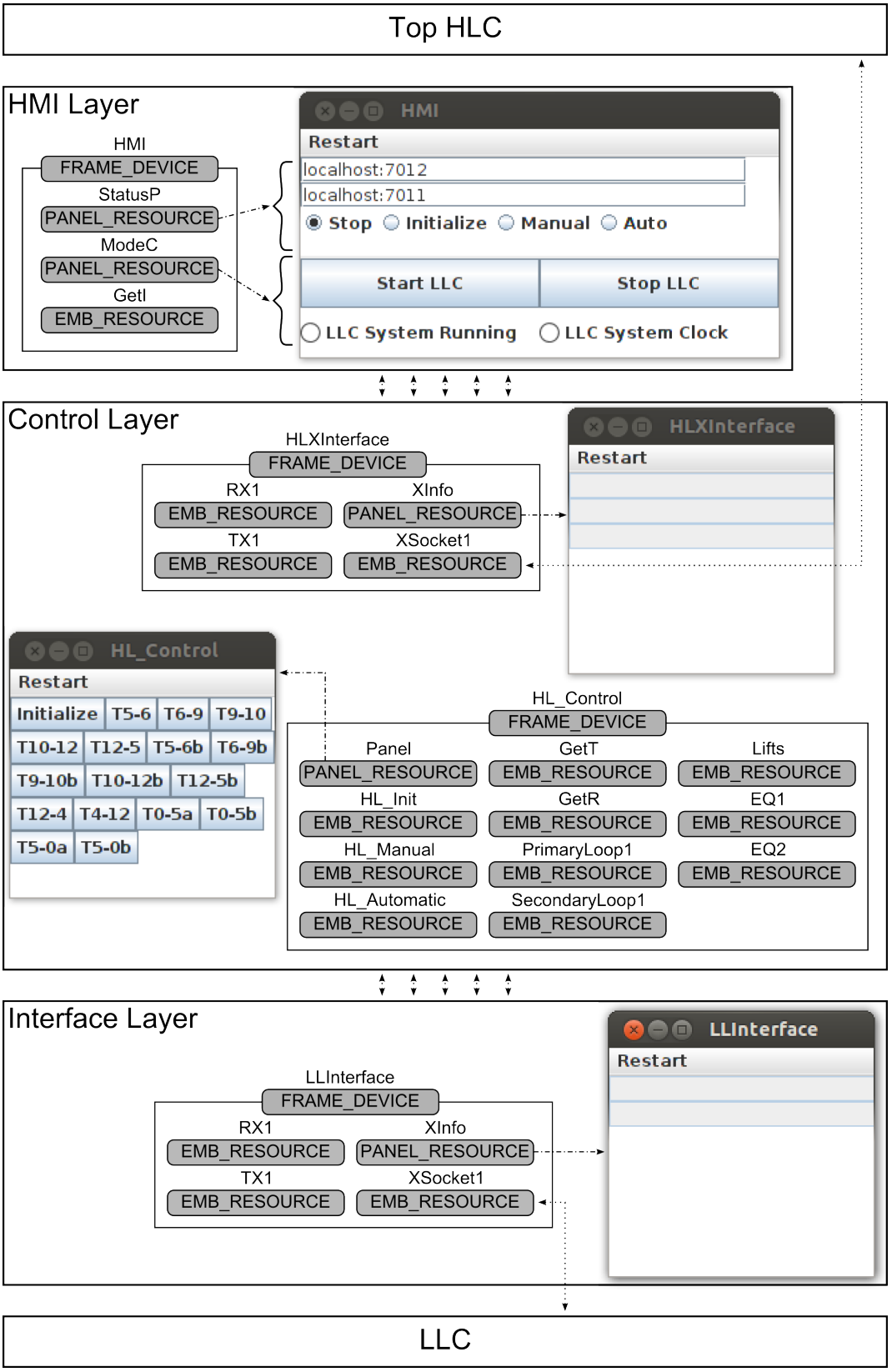


Figure 7.1: Function block controller layout

The HL_Init resource (see section B.1.2) sets initial values of the control network. The primary initialization to be done, is the setting of the number of tokens that are present in each position in the Petri-Net. A program structure is needed for this initialization task that loops through the positions and sets their initial contents. Creating such a structure with the standard IEC61499 function blocks is not as trivial as other common programming languages (see section B.2).

The HL_Manual and HL_Automatic resources (in HL_Control) have function blocks that subscribe to function blocks in ModeC. Switching to manual and automatic mode on the HMI GUI publishes events that are forwarded by HL_Manual and HL_Automatic, to the HLXInterface device. Two embedded resources (GetT and GetR) subscribe to the read and transmit packets from the LLC interface device, process them and publish the information for use in the Petri-Net elements. Issues have been encountered where events are split and are discussed in section B.3.

The remaining resources in HL_Control contain the Petri-Net and facilitate the control thereof (see section 7.4). Two main types of function blocks exist in the PrimaryLoop1 and SecondaryLoop1 resources, namely FBC_Pos and FBC_Trans (see figure 7.2). The inter-connection of these resources determines the layout of the Petri-Net. All events published by the controller on addresses "`PIDx_PXIb`", "`PIDx_PXIs`", "`PIDx_PXIk`" and "`PIDy_PXI`" are subscribed to by FBC_Pos and FBC_Trans (`x` and `y` are the position and transition IDs respectively).

Each interface device has an XSocket1 resource that subscribes to TX and sends any packets to the socket function block and publishes any packets received from the socket function block. The functions of TX1 and RX1 are to multiplex and de-multiplex packets. XInfo serves as the device GUI and also subscribes to publishers in XSocket1 that publish the status of the socket and incoming and outgoing packets.

7.3.3 LLC-HLC Communication

As discussed in section 5.3.4, TCP/IP is selected for communication between the LLC and the HLC. An open-source library, LibNoDave (available at `http://libnodave.sourceforge.net/`), is used to establish communication between a PLC and a computer. ProDave, supplied by Siemens, is a commercial alternative to LibNodave. LibNoDave supplies the user with Linux and Windows C, C++, C#, Delphi, Pascal, Perl and VB(A) source code. A pre-compiled LINUX shared library, a Win32 and a .NET support dynamically linked library and a Pascal interface unit, are available. LibNoDave is also packaged in a Java class. By including these libraries in programs running on a computer with Ethernet support, the programmer can, among others, read and write bits and bytes on the PLC. LibNoDave is used in the interpreter, defined in section 5.3.4, as the interface to the PLC (the LLC).

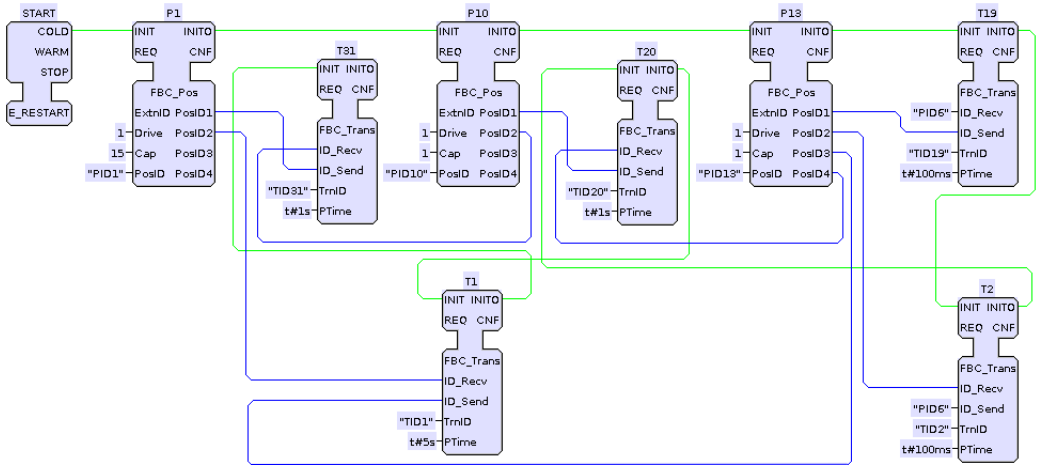


Figure 7.2: Petri-Net function block network

The interpreter is implemented in two different ways. Initially the interpreter is built as a separate program running in C++. The interpreter runs a socket server on which a socket client class of the HLC (ClientLLC function block in figure B.3(a)) and the Socket 1 objects in figures 8.1 and 9.1) can log onto. The second implementation of the interpreter includes the interpretation unit and the LibNoDave in the ClientLLC function block and the Socket 1 objects. Since all the HLCs are implemented in Java, the LibNoDave Java class is used in the second implementation of the interpreter. A major communication speed issue is discovered and discussed in section 10.1.2.

7.4 Reconfigurable Control

The Petri-Net model was used to implement reconfigurable control with function blocks. The holarchies in section 6.2 are sub-layer holarchies in operational holons of the IEC61499 controller. The operational holons are represented by position and transition function blocks. Position and transition function blocks are connected to form a Petri-Net model that controls the motion of tokens (pallets). The ability of building Petri-Nets with function blocks in a graphical environment may enable an operator to reconfigure or maintain a manufacturing system, because of the programming simplicity and intuitiveness.

As seen in figure 7.2, the position function blocks are of type FBCPos, while the transition function blocks are of type FBCTrn. Start position function blocks are connected to end position function blocks with an intermediate transition function block. The position function blocks contains the parameters `Block`, `Drive`, `Cap` and `PosID`. `Block` takes the position ID of other positions, that may block this position, in the form of `PIDi`, where `i` is an integer that specifies the unique ID number for each position. `Drive` is used to indicate

whether the position is time driven (0), or sensor driven (1) and the number of pallets that the position can take is indicated in `Cap`.

The function block's position ID is indicated in `PosID`, also in the format `PIDi`. Four copies of this ID are then made and outputted to the outputs `PosID1` to `PosID4`, for connection with transitions. The transition function blocks have `fromPos` and `toPos` inputs to which the start and end position IDs are connected respectively. Note that the `PosID` can be connected, either with a data connection, or text in the format `PIDi`. The position ID string is used for the addresses of publishers and subscribers with which transition and position function blocks communicate with each other.

Chapter 8

Object-Orientated Controller Design

8.1 Design Strategy

This chapter discusses the design of the object-orientated controller for the upper HLC of concept 1 (the function block controller in figure 5.2). As described in section 7.1, the objective of the top HLC is the path control of pallets, which involves the path selection and navigation from one resource to another. Note that the object-orientated controller is also used in the agent-based bottom HLC. Because object-orientated control is used in both control concepts of this project, the strengths and weaknesses of object-oriented architectures are also investigated.

Object-orientated programming includes features such as data abstraction, encapsulation, messaging, modularity, polymorphism, instantiation and inheritance. Code can also be encapsulated and made modular for easier maintenance. Many instances of the same objects can be dynamically instantiated, a useful property when, for example, multiple agents are needed.

8.2 Manufacturing Entity Object Framework Design and ADACOR Architecture

The manufacturing entity object modelling methodology (section 2.4.5) is used in the creation of the object layout of this controller (Zhang *et al.*, 1999). ADACOR holons are used as the foundation on which the manufacturing entity object is modelled and can be seen in figure 8.1. An implementation of the sub-level holarchy in figure 5.6 is shown in figure 8.1, with the communication from the top level holarchy coming through the cell controller's operational holon. Although other objects are used to facilitate communication and manage program states, the core controller objects are the configuration processor,

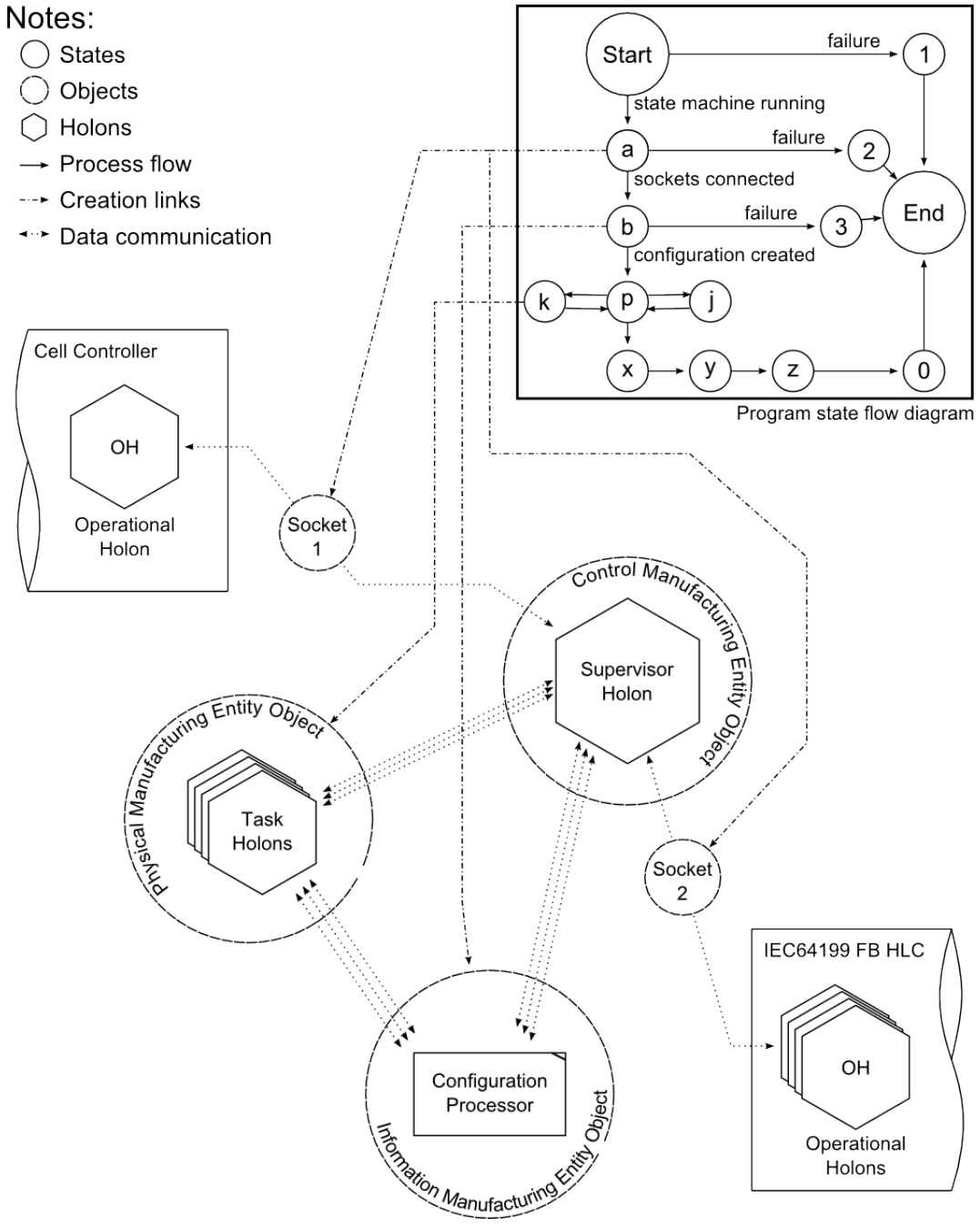


Figure 8.1: Object-orientated controller layout

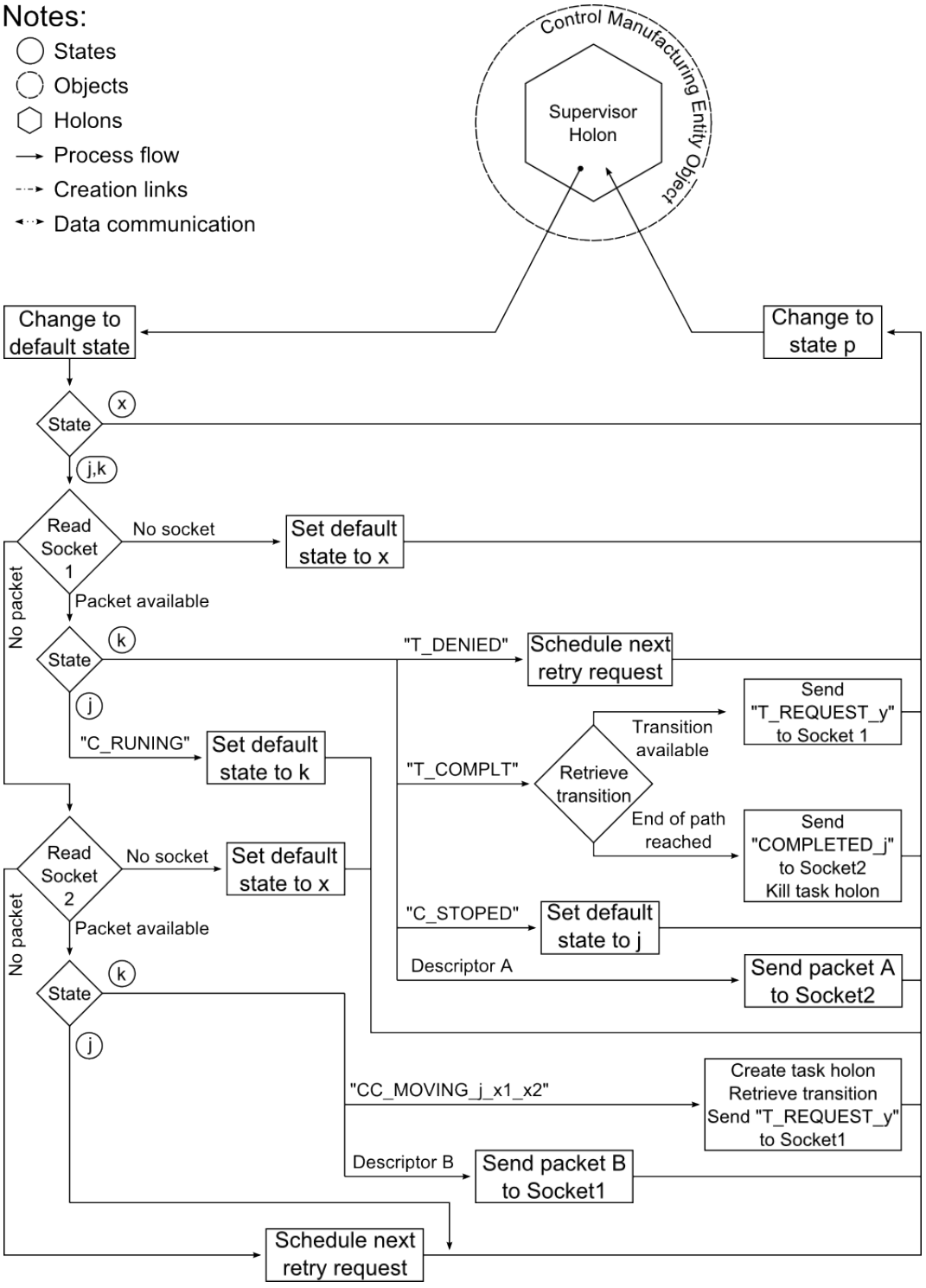


Figure 8.2: Object-orientated controller flow diagram

the task holon and the supervisor holon. The configuration processor object is created to fulfil the role of the information manufacturing entity object. The task holon objects fulfil the role of the physical manufacturing entity object. The supervisor holon is also seen as an object and is considered to be the control manufacturing entity object. As discussed in section 5.4, the operational holons are in a lower control level (the IEC61499 function block HLC in this case) and a communication link exists between the supervisor holon and the operational holons.

The process flow is controlled by a finite state machine (see section 2.4.1), that can be seen in the top right corner of figure 8.1. When the program starts, the finite state machine object is immediately started and state `a` is entered. Note that if the finite state machine start-up fails, the program is exited with an error signal of 1. The same holds for the failures of socket connections (exit error signal 2) and configuration creation (exit error signal 3). An exit error signal 0 indicates correct program shutdown. The basic structure of the main program initialisation can be seen in code C.1.

In state `a`, a menu thread is first started to allow the user to safely shut the program down at any time, while the main program is running. Finally, the socket communication objects are started and the state is set to `b`. In state `b`, the configuration processor object is started. The functioning of the configuration processor object is discussed in section 5.3.4. A default state variable exists that holds the state to which the supervisor holon will change to, from state `p`. Initially the default state is `j`. Figure 8.2 is a continuation of 8.1 and shows the flow diagram of the main program (contained in the supervisor holon). If the configuration processor is successfully created, state `p` is entered and the supervisor holon will change to the default state (the first block illustrated in the flow diagram in the lower part of figure 8.2). If in state `x`, the controller will lead to state `y` (waiting for the menu thread to join the main thread), state `z` and shutdown (exit with error signal 0).

Socket 1 is a connection to the HLC and socket 2 to the CC. The HLC is considered a lower level controller, than the CC, and more real-time. For this reason, socket 1 is first checked for available packets, and secondly, the less demanding socket 2, if socket 1 has no work to be done. If, while reading any of the sockets, the connection is lost, the default state is set to `x` and state `p` is returned to.

In state `j`, packets are only tested for a descriptor "`C_RUNING`", from the HLC, after which the default state is set to `k`. Similarly, if "`C_STOPED`" packets are received in state `k`, the default state is set back to `j`. Task holons are created when a packet `CC_MOVING_j_x1_x2` is received from the cell controller, where `x1` and `x2` are the start and end resources, and `j` is the job number. The holon's first transition is retrieved and the request sent to the HLC as `T_REQUEST_y`, with `y` the transition number.

Section 8.3 described the path table that is used to store weights of all the transitions for each path. If, for a certain path, there are two transitions going

out of one position, the transition with the highest weight will be triggered. In this way a path planner can dynamically direct a pallet through the system, by changing the weights in the path table, in run-time.

If "`T_COMPLT_y`" is received from the function block controller, the next transitions are retrieved and the requests sent to the function block controller. When resource `x2` is reached (the end of the path), a "`COMPLETED_j`" packet is sent to the cell controller and the task holon is closed. Alternatively, if a "`T_DENIED_y`" packet is received from the function block controller, the request is rescheduled to be sent at a later time. Rescheduled requests are attempted to be sent if no packets are received after reading socket 2. Note that some packets received on one socket result in the sending of packets on the other port (descriptor A/B on socket 1/2 result in packet A/B on socket 2/1). This is used to pass the hardware interface messages, described in section 5.3.5, through the controller.

8.3 Path Control

The objective of path control is to determine a path from one resource to another and trigger the transitions in a sequence to accomplish the transportation of a pallet on this path. The functioning of this objective is described in this section. A table with p rows by q columns is used, where p is the number of transitions and q is the number of combinations of paths between the resources. The value of q can be calculated as $q = r_s(r_s - 1)$, where r_s is the number of resource positions (positions with TYPE R in section A.2).

Initially the table is filled with zeros. Ones are then entered to set specific transitions active for each path. The path table is further modified by a path planner and scheduler, which can enter any value between, and including, zero and one. A user or external controller can send a command to the HLC, requesting the transportation of a product from one resource to another.

The controller will start at the position of the starting resource, find all transitions going out of that position and trigger the one that has the highest value in the path table. This process will be repeated until the product has reached the destination resource.

Chapter 9

Agent-Based Controller Design

9.1 Design Strategy

The HLC of concept 2 has the same objectives then that of the HLC of concept 1 (figure 5.2), described in sections 7.1 and 8.1. Although both objectives, traffic control and path control, are discussed in this chapter, the traffic control objective are implemented, in the lower HLC (section 8.1), with object-oriented control, while the path planning objective are implemented with agent-based control. The motivation for this is explained in section 9.2.

9.2 ADACOR Reference Architecture

The ADACOR holon architecture provides a base on which agents can be organized and is used to allocate agent responsibilities. An implementation of the sub-level holarchy in figure 5.6 is shown in figure 9.1, with the communication from the top level coming through the cell controller's operational holons. The supervisor agent manages connections to the cell controller and operational holons. The supervisor agent also instantiates and maintains the configuration processor. Task agents are created by the supervisor agent and employed when jobs are received from the cell controller. As discussed in section 5.4, the operational holons are in a lower control level (the LLC in this case) and a communication link exist between the supervisor holon and the operational holons. This communication link is via the interpreter that is defined in section 5.3.4.

A major difference between object-oriented HLC (path controller) of concept 1 and the agent-based HLC, is the higher autonomy of the task agents. Instead of the supervisor agent initiating all control, as the case with the object-oriented HLC, the task agents can receive jobs and initiate the tasks to complete the job. The operational holons, however, are implemented with object-oriented control, in order to rather focus agent-based implementation on the path control of the supervisor and task agents.

Note that an alternative is to implement the operational holons as agents as well. The reasoning for this is that the function of the operational holons, and communication between operational and task holons are very elementary (the operational holons have less than 10 variables, of which most are boolean variables). With a number of operational holons in the order of 100, the worth of the added agent resources, while objects are sufficient, is questionable, and is considered to result in an over designed controller.

Although the full advantage of agent-based holons is not currently explored (tools like ontologies or advanced behaviour structures), the implementation of a path planner will use the powerful communication and autonomous capabilities of agents to a greater extend. The path planner will have to use information from all the agents to plan the path of a task holon, using the potential autonomous communication and responsiveness abilities of agent-based control.

9.3 Implementation

The controller is started by instantiating the supervisor agent. Every agent runs in its own thread and executes a setup method, illustrated in (figure 9.2), on start-up. Agent behaviours are initiated in the setup method, and are used to manage the duties of the agent. The setup method of the supervisor agent creates the GUI and a list of task agents. The GUI manager behaviour is then started to manage communication sockets and start the rest of the behaviours.

9.3.1 Graphical User Interface

The GUI is started in the supervisor agent's setup method and creates the graphic window seen in figure 9.3. The GUI manager behaviour further manages all user inputs by reading the GUI state and translating it into controller states and actions. IP addresses and ports to the cell controller and the LLC are set and socket connections are be made by clicking the "Connect" buttons. The GUI manager behaviour monitors the socket connections, stopping automatic control if any connection is down and warning the user. After connections are established, the user must also load the configuration file by selecting "Load configuration" from the Start menu and selecting the configuration file from the appearing file dialogue box. The log textbox logs the configuration file that is loaded, and continues to log other information. The controller is now ready to enter manual operating mode and this can be done by clicking on the "Manual" button.

When the controller is switched to manual mode, the socket communication behaviour is started and commands can be sent to, and received from the cell controller and LLC. This includes, turning the hardware on or off with the "On" and "Off" buttons or selecting commands from the "Commands" menu, and

sending them. Note, automatic commands (e.g. commands to move pallets), received from the cell controller, will be placed in a queue and only executed when the controller is switched to automatic mode. By selecting "fillPlace" from the "Commands" menu, the command "`fillPlace,1,15;`", will appear in the textbox below the log textbox. Upon pressing the log enter button, "Enter", next to this textbox, the command will be sent to the configuration processor and logged. This command initialises position 1 with 15 pallets. As soon as the "Auto" button is pressed, the controller will switch to automatic control mode and start reading and executing commands from the cell controller.

9.3.2 The Process Flow Mechanism

Agent-based control uses agent behaviours as the process flow mechanism. The first behaviour is started in the setup method. A second behaviour is then started in the first behaviour to do core tasks. When the core tasks are finished, the second behaviour is closed.

All the behaviours, in a pool of active behaviours, are executed in a sequence that is continuously repeated. If the process gets stuck in one behaviour, none of the other behaviours are executed. A problem is encountered when, instead of creating the agents beforehand, it is attempted to rather create an agent on demand, inside of a behaviour, and wait until the agent is available to allocate a job to it. It is discovered that the agent could not be created since the behaviour never ended to allow the message to be sent to the agent management system. This is one weakness of the behavioural process flow mechanism of agents.

9.3.3 Duty Management

Duty management is conducted in the socket communication behaviour. Similarly to the top HLC of concept 1 (section 8.2), the lower level socket 1 is checked before socket 2. The configuration processor is updated if any information for the configuration processor is available on the socket. If no packets are available on socket 1, socket 2 is checked.

When a job packet is received on socket 2, the task manager behaviour is created, if it does not already exist, and the job is allocated to a task agent. The job allocation process involves searching for the first available task agent on the directory facilitator (with agent state "waiting"), and sending the job to the task agent with a REQUEST ACL message (figure 9.4(a)). The task agent handles further job tasks by communicating with the task manager behaviour, until the job is completed.

Notice that packets received by socket 1 for the cell controller are sent to socket 2, and packets from socket 2 for the configuration processor are sent to socket 1. This is used to pass the hardware interface messages (section 5.3.5)

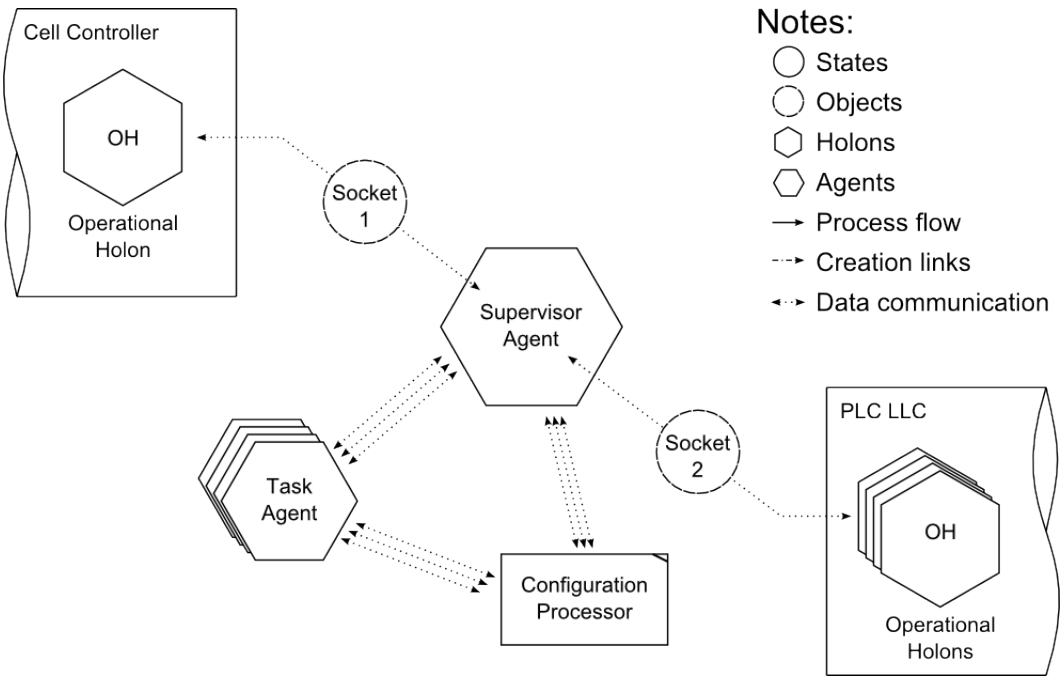


Figure 9.1: Agent-based controller layout

through the HLC. If neither socket 1 nor 2 has packets available, packets in the queue 1 and 2 are written to socket 1 and 2.

9.4 Agent Communication

The agent content language (ACL) is used for agent communication protocols which are shown in figure 9.4. When a job is received from the cell controller, the supervisor agent allocates the job to the first available task agent. This is done with the movement protocol (figure 9.4(b)). The supervisor will look for an available task agent, and forward the job command to the task agent. The task agent then will either agree to take a job and start the transport behaviour, or refuse to take the job.

When the job is allocated, all further communication is initiated by the task agent. Task agents send a call for proposal message to the supervisor agent, as soon as the task agent's transport behaviour is created. Note that the configuration processor, that is a reflection of the operational holons, is contained in the supervisor agent. The call for proposal message (CFP, figure 9.4(b)) is directed to the operational holons than the supervisor agent, but the supervisor agent handles the communication, as discussed in section 5.4. The supervisor replies on the CFP by proposing all the available transitions and their path weights to the task agent. The task agent can reply with either an ACCEPT_PROPOSAL message, containing the accepted transition

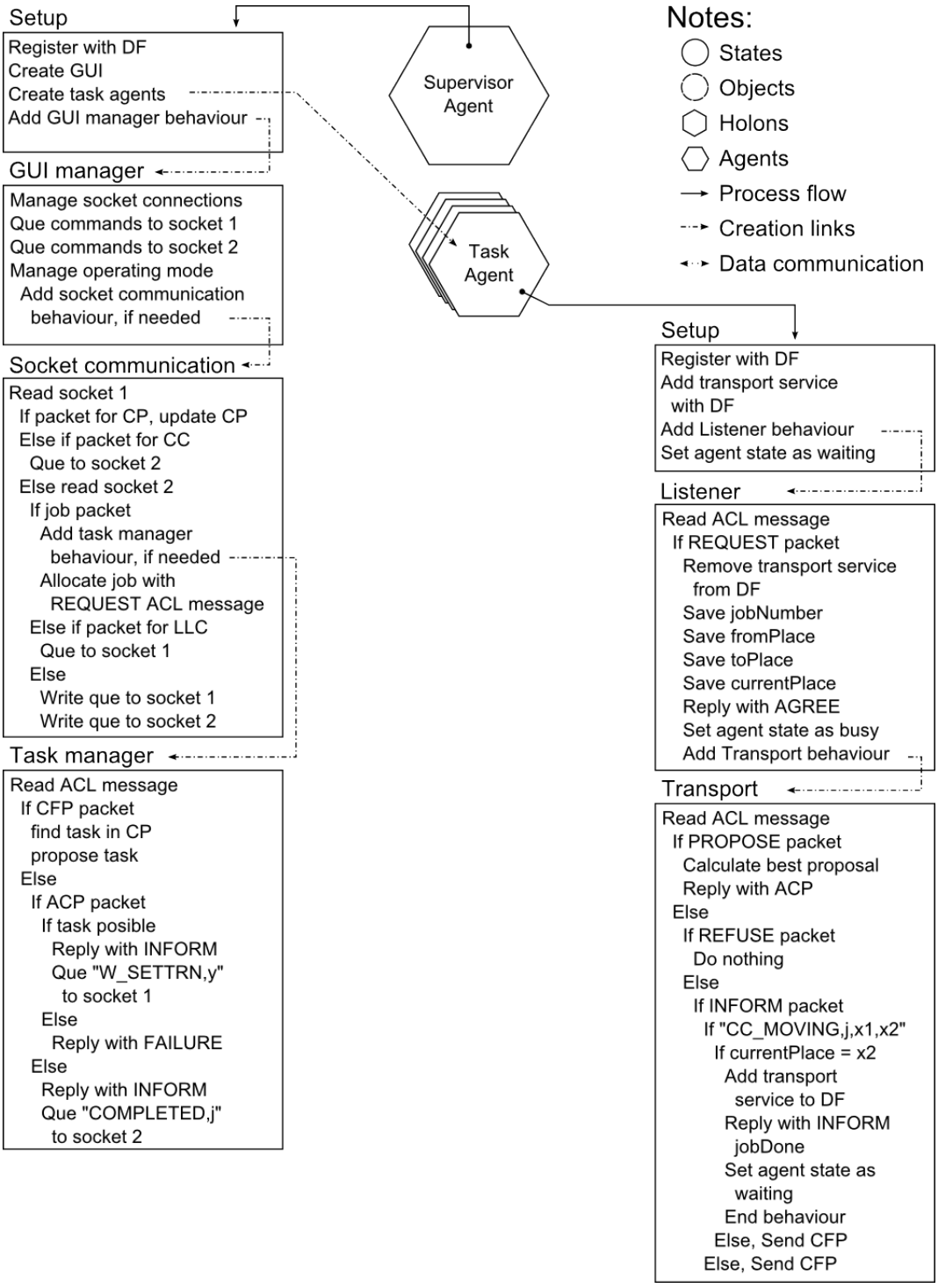


Figure 9.2: Agent-based controller flow diagram

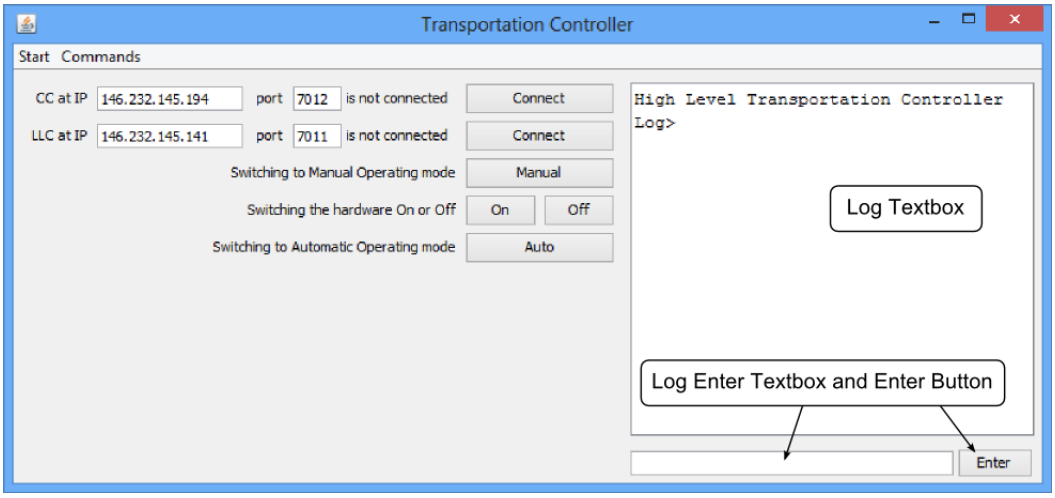


Figure 9.3: Agent-based controller GUI

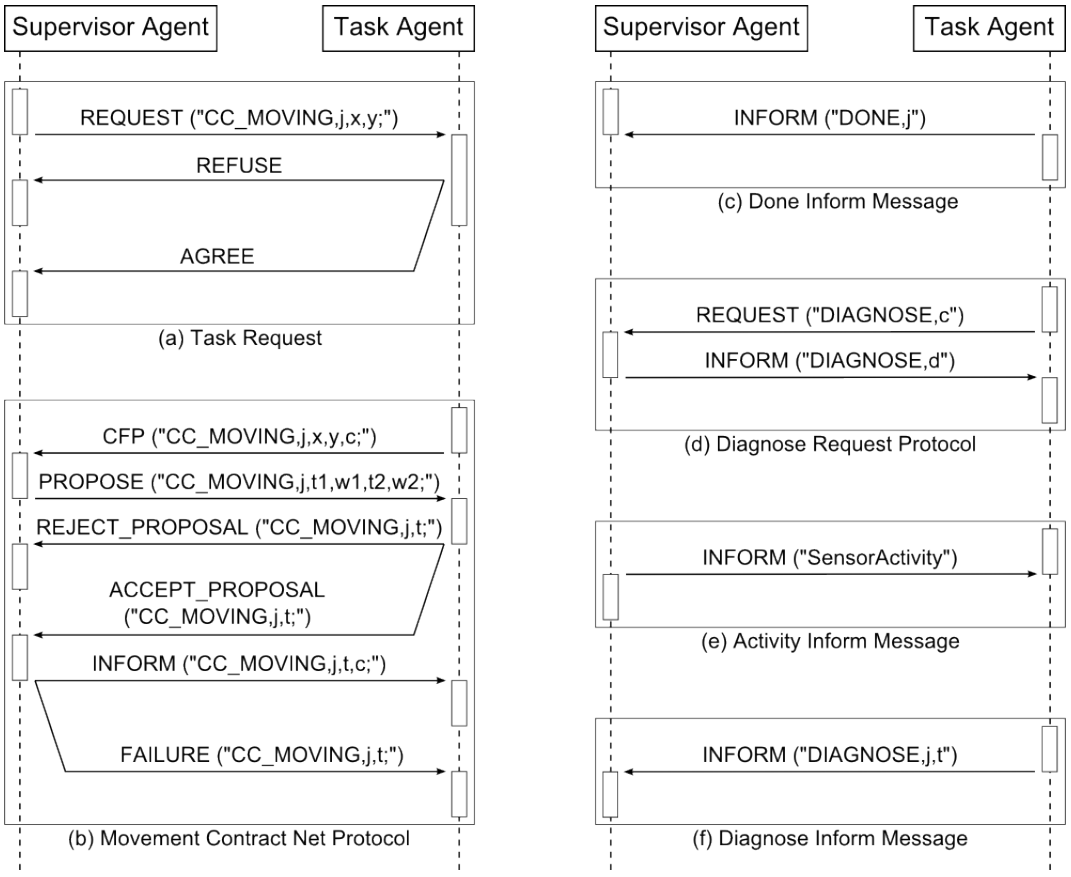


Figure 9.4: Agent-based ACL communication

(with the greatest weight), or a REJECT_PROPOSAL message, if all of the transitions have path weights of zero (see section 8.3 for the functioning of the path control).

Upon receiving an ACCEPT_PROPOSAL message, the supervisor agent will first check the status of proposed transitions and trigger the transitions if possible. If the transitions can be triggered, an INFORM message will be sent to the task agent. If the status report shows that the transition cannot be triggered, a FAILURE message is sent to the task agent and a transition-time-out timer is started. When the transition-time-out timer expires the task agent will inform the supervisor agent to send a diagnostics report to the cell controller (figure 9.4(f)). The transition-time-out timer will be cancelled if the task agent has completed a transition.

Whenever the supervisor agent realises any change of sensor variables in the operational holons, an INFORM message, containing `SensorActivity`, is sent to all active task agents (figure 9.4(e)). Any agent for whom the last transition attempt has failed will, upon receiving this INFORM message, send another CFP message to the supervisor agent (figure 9.4(b)). The task agent can at any time request a diagnosis on specific operational holons, with the diagnosis request communication (figure 9.4(d)). When a task agent reaches the end of its path, it will request a diagnosis on the operational holon of the last position. If the diagnosis shows that the pallet has reached its final position, the task agent will send an INFORM message containing `DONE,j` (where `j` is the job number), and reinitialises itself (figure 9.4(c)).

Chapter 10

Evaluation and Discussion

In this chapter, the reconfiguration strategy, and how well the various control architectures subscribe to the requirements in chapter 4, is evaluated. The reconfiguration strategy of the controllers is first evaluated, followed by a reconfiguration performance evaluation and comparison of the various controllers.

10.1 The Reconfiguration Strategy

The reconfiguration strategy is discussed in section 5.3.1. This is one of many possible strategies and is evaluated by considering three important questions:

1. Is the external modelling method the best solution to the reconfiguration of the conveyor controller?
2. Is the division of the controller into a HLC and a LLC really necessary?
3. Is the use of an IEC61131-3 PLC advantageous?

10.1.1 The External Modelling Method

The external modelling method provides a central data interface by which an external module (a computer aided configuration environment) can be used by the operator to enter configuration information. The required operator configuration information is reduced by increasing controller intelligence (section 3.2). It is the author's opinion, that all reconfiguration strategy possibilities fit on the intelligence-effort scale, that is introduced in section 5.3.1. The external modelling method is located at, what is suspected to be (according to the method in chapter 3), the optimal location of the intelligence-effort scale, for the given specifications, and is thus considered the preferred method.

The external modelling method separates all configuration data from the controller, providing a degree of protection for the controller, from undesired

modifications. Configuration data can be compiled outside the controller and can also be thoroughly tested with simulations. This method may improve modularity and integrability. Because of the ability of off-line testing and simulation, diagnosability is improved and many problems can be fixed off-line.

All configuration information is contained in a human readable file. The operator can read (since it is in a human readable format) and edit the files. The editing ability enables the operator to make small changes without the need of a modelling package. The files can be stored and recalled for later use. The operator can, for example, walk to the manufacturing cell with an USB flash-disk, copy the configuration file on the cell's computer and run the controller with the new configuration. Configuration files can also be created in a different city/country and send to the operator in the factory. This functionality may support the availability characteristic of HMSs, because of the mobility of configuration information.

Using the external modelling method may be more vulnerable to configuration faults that might enter in at the reconfiguration stage, and cause system damage. In order to successfully execute this strategy, a good modelling method is needed which can take all the transportation system situations into account.

10.1.2 The HLC/LLC Division and the Use of PLCs

Dividing the controller into two layers provides a low level control in which the IEC61131-3 can be implemented, without degrading the intelligence of the controller in the higher level. The controller becomes more modular since it is divided into two modules that can each be maintained separately. PLCs are widely used and proven in the industry, while IEC61131-3 is the common programming language standard for PLCs and are highly supported by industry. For these reasons, the use of IEC61131-3 in the LLC increases availability of the control system.

The disadvantage with the separation of the layers is that an additional communication interface is needed between the layers. The management of any communication interface is a significant task that must be additionally undertaken for each connection. The HLC runs on a computer and the LibNoDave library is used to communicate with the PLC. If a platform independent language like Java is used, the HLC has the advantage to work on any platform. The major disadvantage is usually that such communication is much slower than platform specific languages like C++. In the case study, using the Java LibNoDave library for computer to PLC communication proved to be too slow for proper functioning of the controller and the C++ library was used instead. Note that this extreme speed difference might only be specific to the LibNoDave library or operating system but remains an issue to be considered.

Another disadvantage of the layer separation is that one of the controllers may fail, resulting in a synchronization loss that must be managed. Safe start-up and shutdown can be obtained by implementing the HLC as an assisting layer, above the LLC. The HLC can then attempt restoration after the failure. Disturbance handling and robustness may be supported by this property. The development of a reconfigurable IEC61131-3 controller requires the extensive use of indirect addressing. Although IEC61131-3 offers highly intuitive ladder logic and function block programming languages, they do not fully support the use of indirect addressing and the data scanner functions on the PLC had to be written in the statement list language (a low level, assembler type language). Programming in statement list language requires a high programming skill level.

10.2 Function Block and Agent-Based Control

This section describes tests that were conducted on each of the two controller variations. The response of the controllers was then evaluated and compared. The comparisons were used to suggest the most promising architecture (or combination of architectures and their usage) for the controller of a RMS.

10.2.1 The Functional Operation Test

The normal operation of each controller was tested. The first of these tests were the successful transportation of one pallet through the conveyor system. This test showed that all actions were correctly linked to the transitions and that all sensors were assigned at the correct positions. It also showed that the path weights were set-up correctly. The second test was the hardware interfacing. Bits that were set on the LLC of one controller had to be detected by the interpreter and sent through the HLC, to the cell controller. The cell controller had to redirect the signal back to the LLC of the interfacing sub-system. Two such signals had to be sent back and forth between the interfacing sub-systems and had to be robust. A third test was the transportation of multiple pallets to test the traffic control abilities of the controller.

10.2.1.1 Single Pallets

The function block controller, as well as the agent-based controller, completed the basic transportation test successfully. Pallets can be moved on paths from one resource to the next if the HLC receives a transition command from a cell controller. Pallets can be moved from positions that have pallets available, to positions that have space to receive pallets. After the transition hold-time expires (section A.2) or a sensor detects the pallet's presence, the position is filled with a pallet and can give a pallet to the next position. If blocking posi-

tions are filled, the blocked positions are blocked. Some issues with IEC61499, were however identified:

- It was found that programming of common functions (e.g. looping structures, string operations, etc.) with a standard set of function blocks, can quickly result in a complex function block network.
- Function block communication was found to be difficult. A great deal of effort went into the correction of non-operational communication.
- It was found that strong event-based execution programming is not always possible with a standard set of function blocks. Events may have to be split, and delay function blocks used, in order to obtain correct execution.

Passing a hardware interface messages (section 5.3.5) from the LLC to the cell controller, and back to the LLC, appeared to be a challenging task with the function block controller. Using function blocks to decompose received signals, determining the publisher address, passing it to another function block and recombining the signal, results in complex function block networks. The correct functioning of publishing and subscribing of function blocks requires a high level of function block programming expertise.

The hardware interface messages for the function block controller was not operational and testing was done by manually adding and removing pallets from the conveyor system. The hardware interface messages for the agent-based HLC and the object-oriented controllers were easy to implement since the incoming signals were simply copied to the outgoing message queue, within the same holon (agent or object).

10.2.1.2 Multiple Pallets

Both controllers could handle the movement of multiple pallets on the conveyor, but problems were usually encountered when the pallets came close to each other. Pallets directly following each other presented the most problems to the controllers. Crashes usually occur as soon as pallets approach positions that used the blocking functionality.

The function block controller is particularly weak with handling the blocking functionality. Since a position must be checked if it is blocked by another position, the blocking position must first be queried. This results in a back and forth communication and many events that must be joined.

The object-oriented Java programming results in the successful handling of multiple pallets in close proximity, by the agent-based controller. Traffic control of this controller is handled with objects and may be more suitable where large numbers of holons are tested.

10.2.2 The Reconfiguration Test

A full system reconfiguration was conducted by changing the conveyor configuration from configuration A to configuration B (figure A.4). This involved moving a parallel conveyor to the other side of the main conveyor, and reconnecting the actuators and sensors to the PLC. All configuration changes, that include updating the transition, position, and action configuration tables, as well as the descriptor files and the path selection weights, were done manually. Although the control system was designed to allow these configuration changes to be automated by a configuration environment, with minimal human effort, the reconfiguration process could be thoroughly investigated by making all the changes manually.

Manually changing the transition and position tables are logically simple, but humanly work intensive, and could be automated in a configuration environment. The changing of the action table needs more intelligence, but automating it is possible using an optimisation algorithm. Automating modifications of descriptor files can be done with the information supplied from the transition and position tables, and additional elementary settings. The function block controller requires an additional modification of the system file that is used to create the position and transition Petri-Net function block network, as shown in figure 7.2. This needs an additional compiler to compile the system files (section 7.1) for the function block controller, from the configuration environment.

Agent-based and function block control offers distributed control (control distributed over various hardware systems), agreeing more with the holonic paradigm than that of a pure object-orientated architecture. The use of objects to control the lowest level of control in the HLC (the traffic controller, same level as that which the function blocks are used), is preferred over agent-based control. The reason for this is that objects provided sufficient capability for this control and an agent would have resulted in unnecessary complexity.

The feasibility of the use of agents must be carefully considered. Agents add a high level of functionality, but accompanying complexity. Section 9.3.2 shows how even the creation of agents is sensitive to program flow. But the use of agents in the upper control level of the HLC is preferred over object-orientated control. This is because a path controller must be autonomous, since it must enable path optimization, robustness and disturbance handling. Agent-based control is more autonomous than object-orientated control.

Diagnosability was also tested by disabling a proximity sensor, causing the transportation system to deliver a diagnosability signal to the cell controller (protocol f in figure 9.4). The diagnosability signal includes the job ID and an error code to signify the missing sensor confirmation. The same communication complexity issues, that hinders the hardware interface message delivery in section 10.2.1.1, caused problems with the function block controller. The agent-based controller did, however, deliver reliable diagnosability signals.

Chapter 11

Conclusion

The aim of this thesis was to develop the control for a reconfigurable transportation system, which can be used as part of a RMS. Two control software architectures were especially of interest. These are IEC61499 function block control and agent-based control. A conveyor system was chosen as the transportation and served as the case study.

In order to control a system reconfigurably, a more heterarchical or distributed architecture, rather than the common centralized or strong hierarchical architectures, must be adopted. This is necessary to ensure that the reconfiguration effects stay localised at the reconfigured sub-systems. The smaller the effects of reconfiguration are, the less time and costs are involved to do a reconfiguration. This is where the paradigm of holonic manufacturing systems are used to achieve architectures that are hierarchical in nature, but have only localised effects in the case of a reconfiguration.

A major question that must first be answered, is the validity and level of investment toward developing a system to be reconfigurable. A method was developed to determine the target autonomous reconfiguration intelligence level of the system. The method uses parameters that are obtained from the particular application and suggest a design intelligence level that can be used to guide the controller design. In the case study, the target intelligence level was achieved by modelling the conveyor with positions, transition and actions and compiling the model into a configuration data block. The configuration data block is then downloaded onto the controllers. Although the IEC61131-3 standard, used in PLCs, does not enable reconfigurable control, its use was considered important since PLC's are still the preferred controllers in the industry.

The IEC61499 function block control revealed some implementation issues and was found to be difficult to develop. The issues may mostly be attributed to the immature development environment of FBKD. IEC61499 function blocks are more intuitive than conventional programming languages, and may allow operators to change the controller, with little training. Agent-based control are developed in proven Java integrated development environ-

ments. Although agent-based development requires high level programming skills, the Jade platform does offer tools that can further assist development. Agent-based control provides powerful control, above that of IEC61499 function blocks, but may require more resources. With the development issues that are still prevalent in IEC61499, and the fact that the controller can be autonomously reconfigured from a model, the agent-based control is preferred.

By downloading the configuration data block onto the PLC, reconfigurable control was achieved with the PLC. The reconfiguration of both the IEC61499 function block and agent-based controller was easily done. The agent-based controller performed better than the IEC61499 function block controller. Intra-cell synchronisation and diagnosability could only be achieved with the agent-based controller. The transportation of multiple pallets did only work to a limited extent. This shortcoming is subscribed to weaknesses in the model of the physical transportation system. Since the model could be created in an external configuration environment, it can be argued that modularity and integrability, as well as human integration and availability was achieved.

Further work is proposed to develop a better model that may allow robust multi-pallet transportation. Deviations had to be made from the ADACOR holonic architecture, since high level artificial intelligence did not fall into the scope of this work. Further work in developing smarter agents may add disturbance handling to the control system, and make it robust.

Appendices

Appendix A

Configuration Schematics and Tables

A.1 Configuration Schematics

The layout of the conveyor system, shown in figure 1.1, can be seen in figure A.1 showing the placement of all the hardware discussed in section 4.1. Figure A.2 shows the Petri-Net of the system and the way its positions and transitions are linked. In figure A.3, the Petri-Net is superimposed on the layout to show where the positions are located on the system layout. Figure A.4 shows the how the system was changed from configuration A to configuration B.

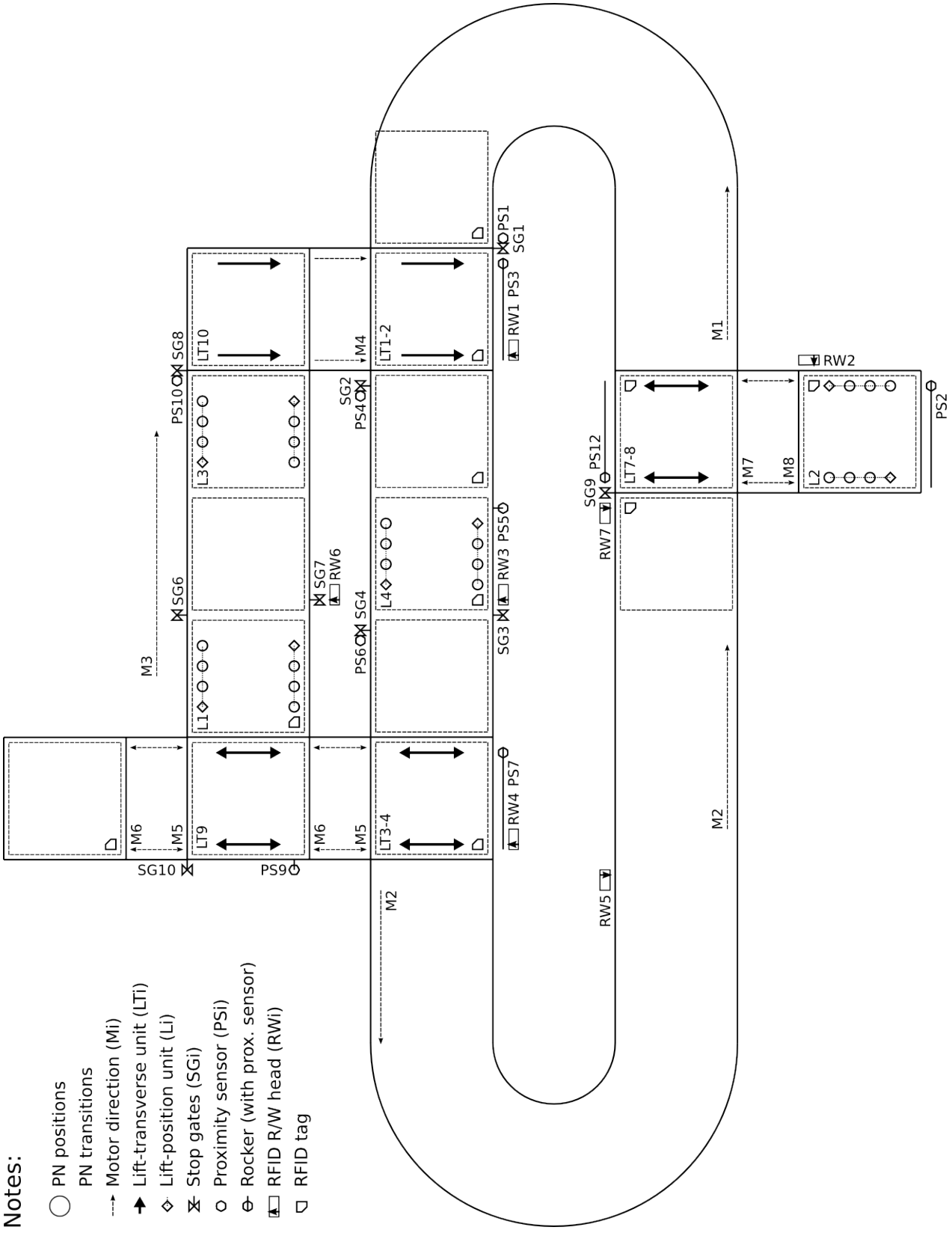


Figure A.1: The conveyor system hardware configuration

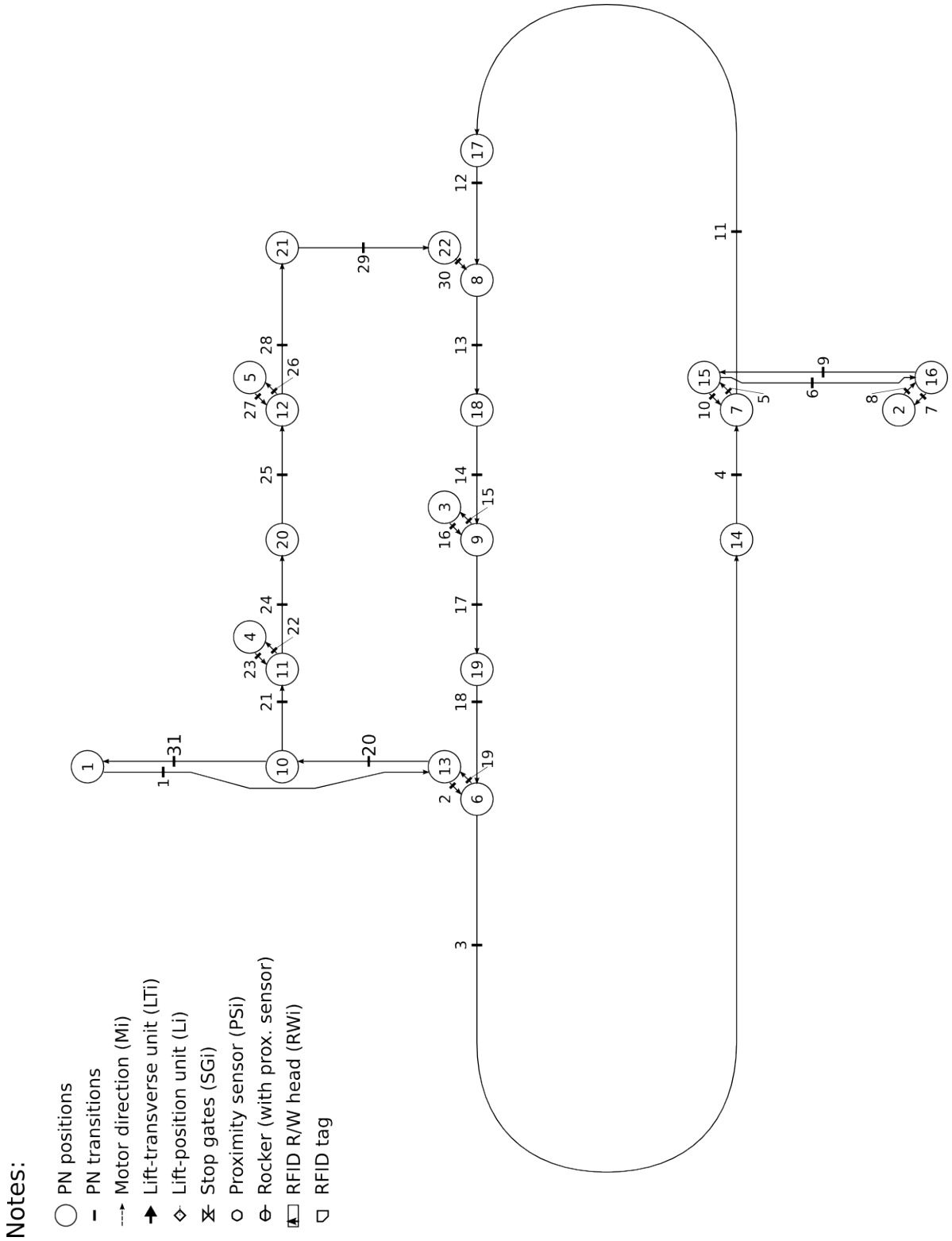


Figure A.2: The conveyor system Petri-Net configuration

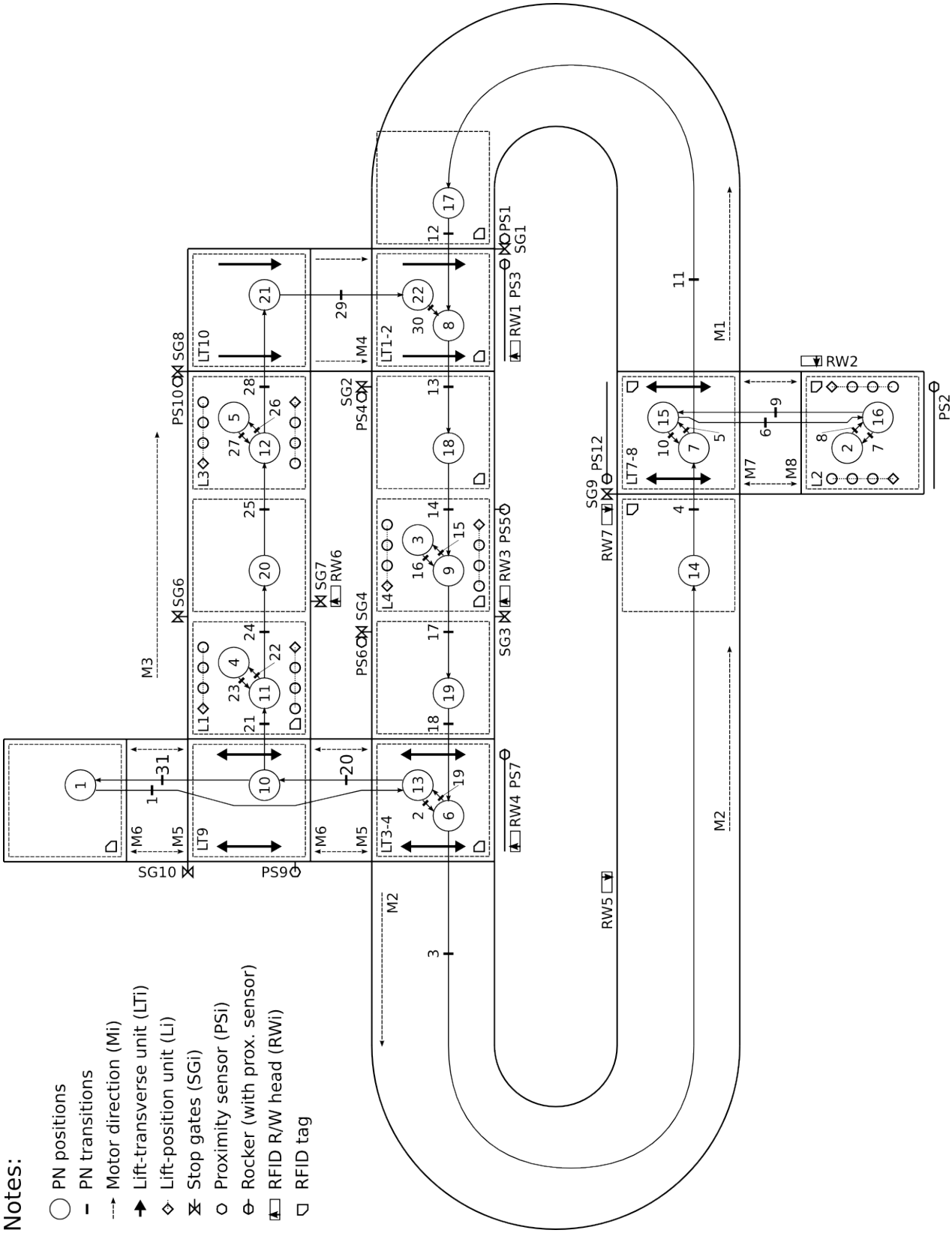


Figure A.3: The conveyor system with hardware configuration and Petri-Net

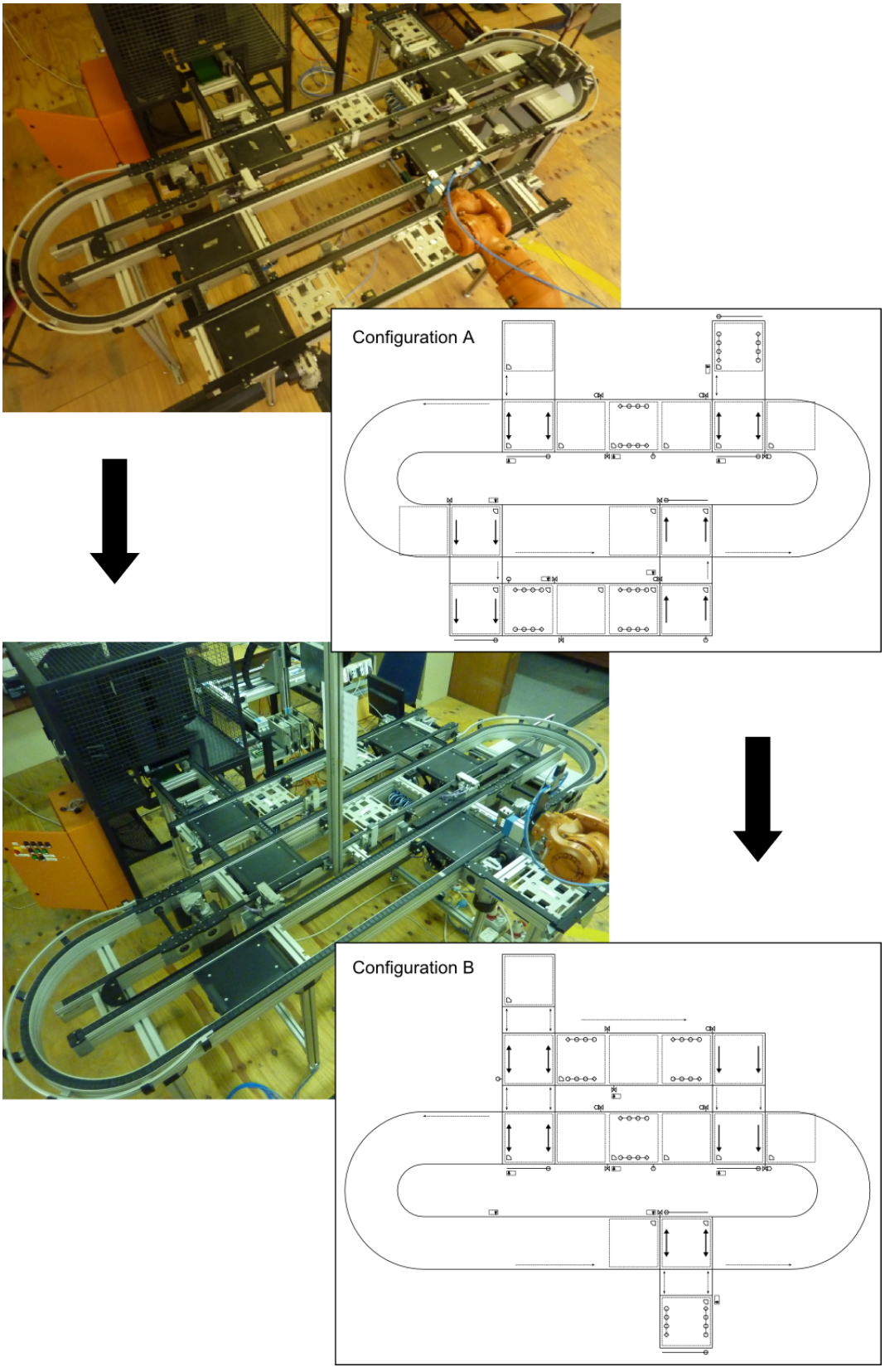


Figure A.4: The reconfiguration test

A.2 Configuration Data Tables

The configuration data blocks, and their fields, are explained in this section. All table entries (rows) contain IDs (`ID`) and text-based names (`NAME`). The ID numbers are used to reference entries, for use in any other entries, in the same or any other table. The NAME only assists human interpretation of the table.

The transition table (table A.1) contains the information needed to describe the Petri-Net structure, as well as detail information of transitions and its related actions. Every entry in the transition table contains the:

- `ID` and `NAME`,
- `STEP`, (a character indicating the step (e.g. `a,b,c`)),
- `SPOS`, (the ID of the transition starting position),
- `EPOS`, (the ID of the transition end position),
- `ADDR`, (an address to trigger the transition),
- `ACT`, (an action to be set (the ID of the action)),
- `!ACT`, (an action to be cleared (the ID of the action)) and
- `TIME` (the transition hold-time value).

The start and end positions are indicated by `SPOS` and `EPOS`, by their respective referenced position IDs. The transition can be triggered by setting (set true) the variable at `ADDR`. This will set the `LEAD` field in action `ACT` true and clear (set false) `LEAD` in action `!ACT`.

Every transition has a hold-time value (`TIME`) that indicates the minimum time in which a transition is allowed to be completed. The hold-time is used to apply a time delay when proximity sensors are not available. The hold-time is also necessary in situations where a sensor may detect a pallet, but the pallet still need time to get into a position. The transition completion is inhibited until the hold-time has expired. In the case where a complex transition is decomposed into a series of discrete intermediate transition steps, the intermediate transition steps are labelled in `STEP`.

The position table (table A.2) contains detail information of positions in the Petri-Net. Every entry in the position table contains the:

- `ID` and `NAME`,
- `TYP` (a character indicating the position type),
- `CAP` (the capacity of the position),
- `BLOCK` (the ID of a position that can block this position),

Table A.1: Transition configuration table

ID	NAME	STEP	SPOS	EPOS	ADDR	ACT	!ACT	TIME
1	T1-6	a	1	13	120.0	10	28	1000
2	T1-6	b	13	6	120.1	13	None	10
3	T6-7	a	6	14	120.2	None	14	10000
4	T6-7	b	14	7	120.3	17	9	2000
5	T7-2	a	7	15	120.4	None	17	10
6	T7-2	b	15	16	120.5	22	25	1000
7	T7-2	c	16	2	120.6	25	22	1000
8	T2-7	a	2	16	120.7	22	None	10
9	T2-7	b	16	15	121.0	18	26	10
10	T2-7	c	15	7	121.1	17	None	1000
11	T7-8	a	7	17	121.2	None	18	1000
12	T7-8	b	17	8	121.3	11	1	10
13	T8-9	a	8	18	121.4	None	12	10
14	T8-9	b	18	9	121.5	24	2	1000
15	T9-3	a	9	3	121.6	None	24	10
16	T3-9	a	3	9	121.7	24	None	1000
17	T9-6	a	9	19	122.0	None	3	10
18	T6-9	b	19	6	122.1	13	4	6000
19	T6-10	a	6	13	122.2	None	13	1000
20	T6-10	b	13	10	122.3	None	27	1000
21	T10-11	a	10	11	122.4	19	None	1000
22	T11-4	a	11	4	122.5	None	21	10
23	T4-11	a	4	11	122.6	21	None	1000
24	T11-12	a	11	20	122.7	None	6	10
25	T11-12	b	20	12	123.0	23	7	10
26	T12-5	a	12	5	123.1	None	23	1000
27	T5-12	a	5	12	123.2	23	None	10
28	T12-8	a	12	21	123.3	20	8	1000
29	T12-8	b	21	22	123.4	12	20	1000
30	T12-8	c	22	8	123.5	11	None	10
31	T10-1	a	10	1	123.6	None	10	10

- ADDR (an address to activate a transition),
- PRES (an address indicating the presence of a token),
- SENS1 (the ID of a sensor action) and
- SENS2 (the ID of a sensor action).

Positions are classified in types (TYP) as resources, junctions or intermediate steps with R, J and I respectively. The capacity of positions (the maximum number of tokens that it can contain) is indicated in CAP. In some cases, more than one position may occupy the same physical space (e.g. at the lift-position units, a pallet can either rest on the conveyor or be lifted). In these cases, a pallet in any one of the positions, occupies all the positions. Blocking is introduced to stop tokens (pallets) to enter such positions. Some positions are blocked by others by artificially filling the position (to its capacity) when the blocking position contains any tokens. The blocking position is indicated with BLOCK. The variable at PRES is set if the position contains any tokens. The variable at ADDR can be used as an alternative transition trigger mechanism from a position perspective. Initially, pallets were transferred by setting the ADDR true. It is considered easier since only a bit of the end position has to be set. It was later found that, only specifying the end position may lead to a transition from the wrong position. Transition triggering (by specifying the transition) was added and used, but both triggering mechanisms are still available. Setting the ADDR true will cause an inbound transition to be triggered if the transition's pre-connected position has tokens available (PRES is true). References to action IDs of sensors at the position are stored in SENS1 and SENS2.

The action table (tables A.3 and A.4) contains detailed information regarding the connectivity of actuators and sensors and their mapping to variables in the hardware interfacing controller. Every entry in the action table contains the:

- ID and NAME,
- TYPE (a character indicating the action type),
- LEAD (an address to the leading variable),
- FOLLOW (an address to the following variable),
- PROP (the properties of the action (a seven bit number)),
- ACT1 (the ID of linked actions),
- ACT2 (the ID of linked actions) and
- DATA (an address to the data variable).

Table A.2: Position configuration table

ID	NAME	TYP	CAP	BLOCK	ADDR	PRES	SENS1	SENS2
1	P0	R	15	None	110.0	110.1	31	None
2	P1	R	1	None	110.2	110.3	32	RW_6
3	P2	R	1	None	110.4	110.5	33	RW_7
4	P3	R	1	None	110.6	110.7	34	RW_2
5	P4	R	1	None	111.0	111.1	35	RW_3
6	P5	J	1	13	111.2	111.3	36	RW_4
7	P6	J	1	15	111.4	111.5	Time	RW_5
8	P7	J	1	2	111.6	111.7	38	RW_6
9	P8	J	1	3	112.0	112.1	39	RW_7
10	P9	J	1	None	112.2	112.3	40	None
11	P10	J	1	23	112.4	112.5	41	RW_1
12	P10-3.2	J	1	5	112.6	112.7	42	RW_3
13	P12	I	1	None	113.0	113.1	43	RW_4
14	P0-5.1	I	4	None	113.2	113.3	Time	None
15	P5-6.1	I	1	None	113.4	113.5	Time	RW_5
16	P6-7.1	I	1	None	113.6	113.7	46	None
17	P6-7.2	I	1	None	114.0	114.1	Time	None
18	P7-8.1	I	1	19	114.2	114.3	48	None
19	P8-9.1	I	1	None	114.4	114.5	49	None
20	P8-9.2	I	1	10	114.6	114.7	50	None
21	P8-9.3	I	3	None	115.0	115.1	Time	None
22	P6-9.1	I	4	None	115.2	115.3	52	None

Actions are classified in types (`TYPE`) as actuators, sensors or links with `A`, `S` and `L` respectively. The variable at `LEAD` is modified by a controller, a sensor or a variable of another action. The variable at `FOLLOW` is an actuator that is to be set equal to the `LEAD` variable. References of action IDs can be entered in `ACT1` and `ACT2` and their variables at `LEAD` will be set or cleared, depending on the value of `PROP`. `PROP` is a seven bit integer that specifies the behaviour of the action. Setting the first bit causes the `LEAD` field to be cleared after one delay cycle of the delay clock (see delay clock, section 6.3). The second and third bits determines whether `ACT1` and `ACT2` will be modified on the rising edge (if changed to true) or the falling edge (if changed to false) of the `LEAD` variable. Whether they are set or cleared are determined by the fourth and fifth bits. The sixth and seventh bits are used to specify whether the `LEAD` and `FOLLOW` variables are located in the input, the output or direct memory areas. The variable at `DATA` stores a value that needs to be sent to, or received from the tokens (transportation units like pallets) and can be used by `RFID` units to read and write information from or to the pallets. Note that the `DATA` and `PROP` fields are omitted from tables A.3 and A.4 since it is not used.

Table A.3: Action configuration table 1

ID	NAME	TYPE	LEAD	FOLLOW	ACT1	ACT2
1	SG1	A	150.0	11.2	12	None
2	SG2	A	150.1	11.3	None	None
3	SG3	A	150.2	12.6	None	None
4	SG4	A	150.3	12.7	14	None
5	SG5	A	150.4	12.2	18	None
6	SG6	A	150.5	12.3	None	None
7	SG7	A	150.6	13.6	None	None
8	SG8	A	150.7	13.7	None	None
9	SG9	A	151.0	13.2	None	None
10	SG10	A	151.1	13.3	None	None
11	EQ1	A	151.2	20.0	29	None
12	EQ2	A	151.3	20.1	None	None
13	EQ3	A	151.4	20.2	28	19
14	EQ4	A	151.5	20.3	None	None
15	EQ5	A	151.6	11.4	None	None
16	EQ6	A	151.7	11.5	None	None
17	EQ7	A	152.0	11.6	26	None
18	EQ8	A	152.1	11.7	None	None
19	EQ9	A	152.2	21.4	21	None
20	EQ10	A	152.3	21.5	29	None
21	L1	A	152.4	10.0	None	None
22	L2	A	152.5	10.1	None	None
23	L3	A	152.6	10.2	None	None
24	L4	A	152.7	10.3	None	None
25	InFeeder	A	153.0	1.7	None	None
26	OutFeeder	A	153.1	1.6	17	None
27	Into PM	A	153.2	1.5	19	9
28	Outof PM	A	153.3	1.4	14	13
29	OutPconv	A	153.4	1.2	11	None
30	None	A	153.5	1.3	None	None

Table A.4: Action configuration table 2

ID	NAME	TYPE	LEAD	FOLLOW	ACT1	ACT2
31		S	130.0	140.0	None	None
32		S	11.1	140.1	None	None
33		S	12.0	140.2	None	None
34		S	109.1	140.3	None	None
35		S	13.1	140.4	None	None
36		S	13.4	140.5	None	None
37		S	14.5	140.6	None	None
38		S	12.4	140.7	None	None
39		S	12.0	141.0	None	None
40		S	13.0	141.1	None	None
41		S	109.1	141.2	None	None
42		S	13.1	141.3	None	None
43		S	13.4	141.4	None	None
44		S	109.1	141.5	None	None
45		S	14.5	141.6	None	None
46		S	11.1	141.7	None	None
47		S	11.0	142.0	None	None
48		S	12.5	142.1	None	None
49		S	12.1	142.2	None	None
50		S	109.1	142.3	None	None
51		S	109.1	142.4	None	None
52		S	12.4	142.5	None	None
53		S	190.0	142.6	None	None
54		S	160.0	142.7	None	None
55		S	180.0	143.0	None	None
56		L	153.6	143.1	17	18
57		L	153.7	143.2	None	None
58		L	154.0	143.3	None	None
59		L	154.1	143.4	None	None
60		L	154.2	143.5	None	None

A.3 The PLC Memory Layout

A map of the PLC memory allocation is shown in table A.5. Memory addresses 50, 52 and 2000 hold word values (16 bit) indicating the number of positions, transitions and actions. Addresses 54 to 100 are used by the transition scanner while 2002 to 2038 are used by the action scanner. Addresses 100 to 200 are used to control the PLC from a HLC. The first 10 bytes from address 100 hold general status bits. For example, the controller can be started by setting bit 100.0 (the first bit in 100). Transition can be executed by setting bits from addresses 120 to 130 (e.g. setting 121.2 will execute transition 11). The transition and action data are stored between addresses 600 to 2000.

Table A.5: PLC memory layout

Mem	Type	Description	bit:0	bit:1	bit:2	bit:3	bit:4	bit:5	bit:6	bit:7
50	W	number of pos								
52	W	number of trans								
54	W	trans scan index								
56	D	trans scan pntr								
60	D	prev pos pntr								
64	D	post pos pntr								
68	D	trans pntr								
72	D	acts set pntr								
76	D	acts reset pntr								
84	D	trans pntr start								
88	D	trans pntr stp								
92	D	pntr-pntr stp								
98	W	the trans stp								
100	10	status bits	strt	run	stop	clk/2	clk	!clk	clkM	clkU
101	B		tOn1	tOn2	aOn1	aOn2				
109	B			set						dum
110	10	pos bits	p1A	p1B	p2A	p2B	...			
120	10	trans bits	trn1	trn2	trn3	trn3	...			
130	10	dummy map								
140	10	sensor map								
150	10	actuator map								
160	10	HWI in-int								
170	10	HWI in-ext								
180	10	HWI out-int								
190	10	HWI out-ext								
200	10	RFID data								
600	n	trans fields								
1000	n	action fields								
2000	W	number of acts								
2002	B	config buffer	autR	a1Eg	a2Eg	a1SR	a2SR	M/I	M/Q	
2004	W	acts scan index								
2006	D	acts scan pntr								
2010	D	lead pntr								
2014	D	follow pntr								
2018	D	effect1 pntr								
2022	D	effect2 pntr								
2026	W	config buffer								
2038	B	calculation bits	lead	folw	clkM	ledT	araP	ledP		

A.4 The Descriptor File

The descriptor file format is defined in this section. Code A.1 shows the descriptor file for the PLC. The descriptor file format is defined to be `DESCRIPTOR,CONTROL,MEM,SKIPI,SKIPN,LENGTH,VALUE,MONITOR`. The `DESCRIPTOR` field holds an eight character command descriptor, presented as the major part of the HLC string command. `CONTROL` is a single character that describes the action with either "`R`", "`r`", "`W`" or "`w`" that indicates read or write, bytes (in capitals) or bits respectively. If `MONITOR` is a non-zero value with `CONTROL` as either "`R`" or "`r`", the number of `LENGTH` bytes of PLC memory area at address `MEM`, will be monitored by the interpreter (keep on testing for a change and send a command to the HLC if a change occurred).

The string-based commands from the HLC have the format `DESCRIPTOR;`, `DESCRIPTOR,INDEX;` or `DESCRIPTOR,INDEX,MODIFY;`. The `INDEX` field indicates the bit/byte offset from the starting address `MEM`. If `SKIPI = 0` and `SKIPN = 1`, `INDEX` will correspond to every second bit/byte, starting at PLC address `MEM`. If `SKIPI = 1` and `SKIPN = 1`, `INDEX` will correspond to every second bit/byte, starting at one bit/byte after `MEM`. `MODIFY` can be included to overwrite `VALUE`, the value to be written to the PLC bit/byte in case of `CONTROL` values of "`W`" or "`w`". The string-based commands that are send back to the HLC have the format `DESC,CHANGE,INDEX;`, where `CHANGE` indicates the value to which the bit/byte at `INDEX` must change to.

Sending the command "`W_SETINN;`" to the interpreter (from the HLC) will set the first bit in byte 160 of the PLC memory. Note that byte 160 is also monitored. As soon as the first bit in byte 160 is set, the interpreter will sent a command "`I_HWFINN,1,1;`" back to the HLC.

Code A.1

```
I_STATUS,R,102,0,0,1,0,2
R_POSTNS,R,110,1,1,7,0,3
R_SENSOR,R,140,0,0,5,0,4
I_HWFINN,R,160,0,0,1,0,5
I_HWFINX,R,170,0,0,1,0,6
I_HWFOTN,R,180,0,0,1,0,7
I_HWFOTX,R,190,0,0,1,0,8
W_SETTRN,w,120,0,0,5,255,0
W_SETPOS,w,110,1,1,7,255,0
W_SETSTA,w,100,0,0,3,255,0
W_CLRSTA,w,100,0,0,3,0,0
W_SETINN,w,160,0,0,1,255,0
W_CLRINX,w,170,0,0,1,0,0
W_SETOTX,w,190,0,0,1,255,0
W_CLROTN,w,180,0,0,1,0,0
W_CLROTX,w,190,0,0,1,0,0
```

Appendix B

IEC61499 Function Block Networks

B.1 Function Block Network Schematics

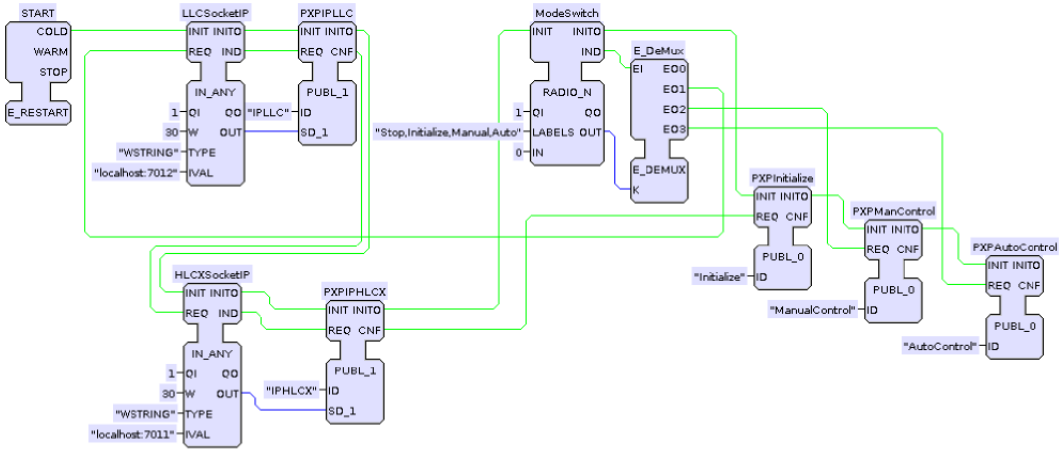
B.1.1 The HMI

Figure B.1(a) shows the ModeC network that allows the user to set the IP addresses of the LLC and external HLC interfaces, as well as the operating mode of the top HLC. The IN_ANY function blocks are the text boxes in which the `IP:port` addresses are entered. The addresses are published to the socket function blocks in the interface devices with the PUBL_1 function block. RADIO_N creates the radio buttons for mode switching. Switching events are published to the interface device of the top HLC of concept 1, with the PUBL_0 function block.

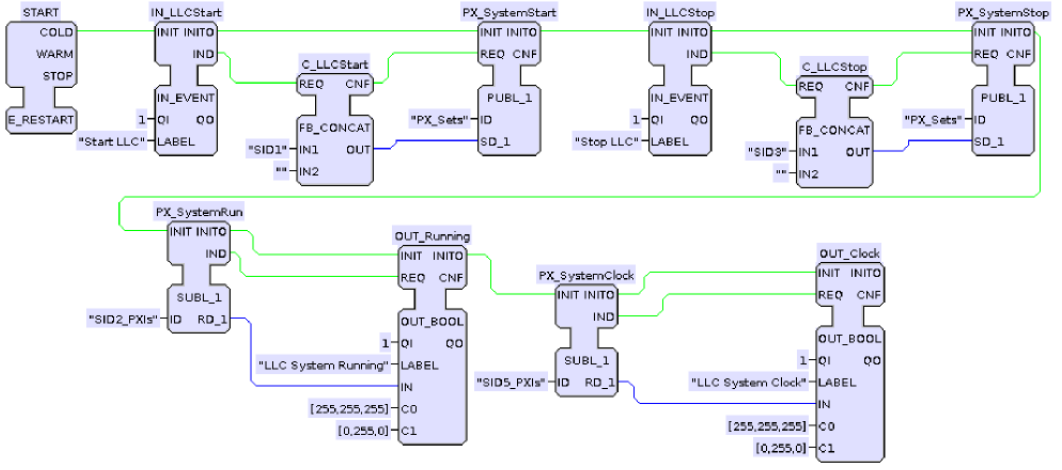
The StatusP resource (figure B.1(b)) enables basic access to the LLC. The LLC can be started and stopped, and an indicator provides limited diagnostics by displaying the system running state. The IN_EVENT function blocks are the buttons that publish commands to the LLC interface device. SUBL_1 function blocks subscribe to a publisher in the GetI resource and display the LLC status with the OUT_BOOL blocks.

B.1.2 The Controller

The HL_Init resource (figure B.2(a)) sets initial values of the control network. Similar to HL_Init, HL_Manual and HL_Automatic (in HL_Control), have SUBL_0 subscribers (PXSManualMode and PXSAutoMode) that subscribe to PXPManControl and PXPAutoControl in ModeC (figure B.2(b) and (c)). Switching to manual and automatic mode on the HMI GUI publishes events that are forwarded by HL_Manual and HL_Automatic, to the HLXInterface device.



(a) The ModeC function block network



(b) The StatusP function block network

Figure B.1: HMI function block networks

Two embedded resources (GetT and GetR) subscribe to the read and transmit packets from the LLC interface device, process them and publish the information for use in the Petri-Net elements. Messages (packets) that come from the LLC interface device will, for example, be "`R_POSITNS_1_13`". The packet format is `d_b_x`, where `d` is an eight character descriptor, `b` is a boolean value and `x` is a number relating to an ID. Because it starts with a `R_`, the LLC interface device (figure B.3(b)) publish it on address "`GotR`", and it is received by PXGetR. Note that C_Dummy is used to convert the unknown type output RD_1 of PXGetR to a string type.

The output of C_Dummy is then sent to three different function blocks. First on the event line is C_Left that separates the first eight characters of the packet ("`R_POSITNS`" in this example). Secondly, a packet is delivered to C_WSAfterK1 that outputs any text after the 11th character ("`13`" in this example). C_AddPID appends the value of `Output` of C_WSAfterK1, to "`PID`" and sends it to C_AddPXIb and C_AddPXIs to further append "`_PXIb`" or "`_PXIs`". These outputs apply the addresses "`PID13_PXIb`" and "`PID13_PXIs`" to the `ID` inputs of PX_Pos and PX_Sens respectively. Thirdly, and next in the event line, a packet is delivered to C_Left2, resulting in "`R_POSITNS_1_`" and then to C_Right2 resulting in "`1_`". This output is then used to emit one of two events, to set `Q` of E_Bool, either true or false, and send to the data inputs of PX_Pos and PX_Sens.

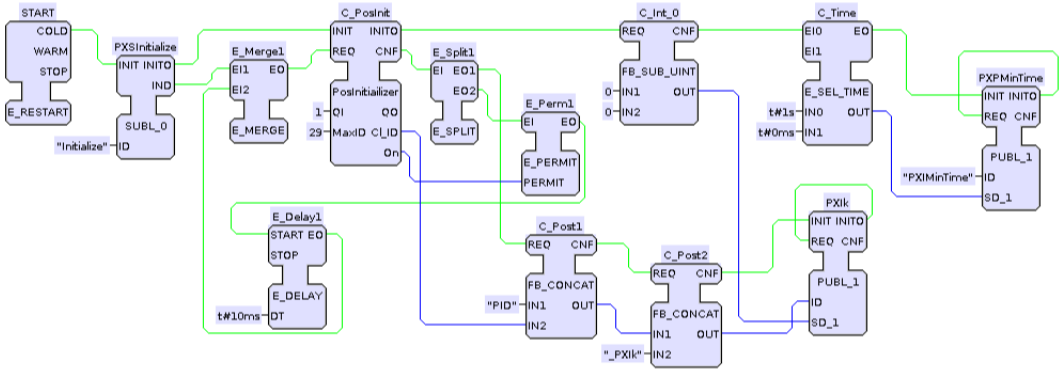
B.1.3 The Interface Device

Each interface device (figure B.3) has an XSocket1 resource that subscribes to TX. It sends any packets to the socket function block and publishes any packets received from the socket function block. The functions of TX1 and RX1 are to multiplex and de-multiplex packets. XInfo serves as the device GUI and also subscribes to publishers in XSocket1 that publish the status of the socket and incoming and outgoing packets.

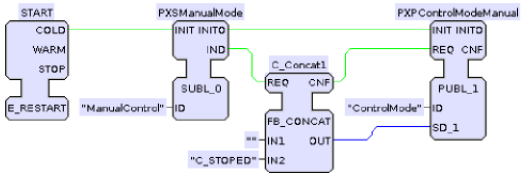
B.2 Looping Program Structures

A program structure that loops through the positions and sets their initial contents, are shown in figure B.2(a). The HL_Init resource has a SUBL_0 subscriber (PXSInitialize) that subscribes to PXPInitialize in ModeC. Thus, as soon as the mode is switched to initialize, PXSInitialize emits an event that is used to start the position initializer loop.

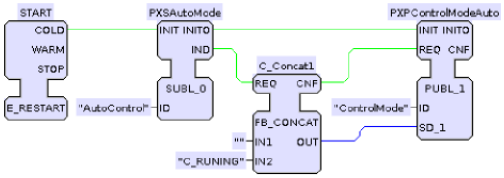
The loop consists of function blocks E_Merge1, C_PosInit, E_Split1, E_Perm1 and E_Delay1. C_PosInit is a custom built function block that counts from 1 to `MaxID` on each incoming `REQ` event. The counter value is outputted on CI_ID and a boolean value (On) remains true until `MaxID` is reached. An event is then emitted from C_PosInit's `CNF` output and split by



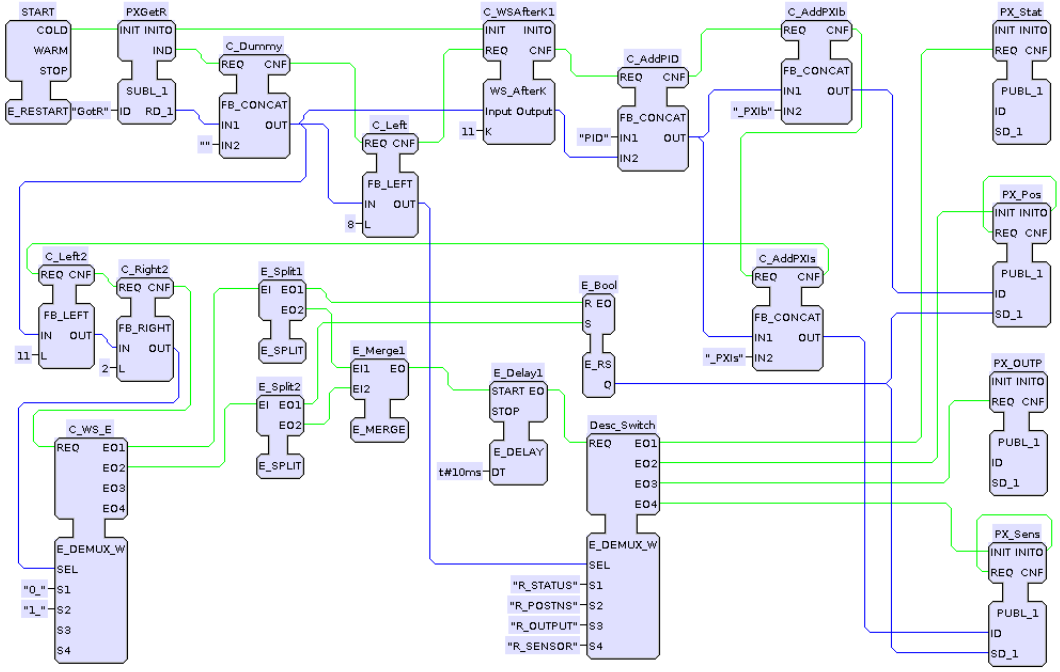
(a) The HL_Init function block network



(b) The HL_Manual function block network

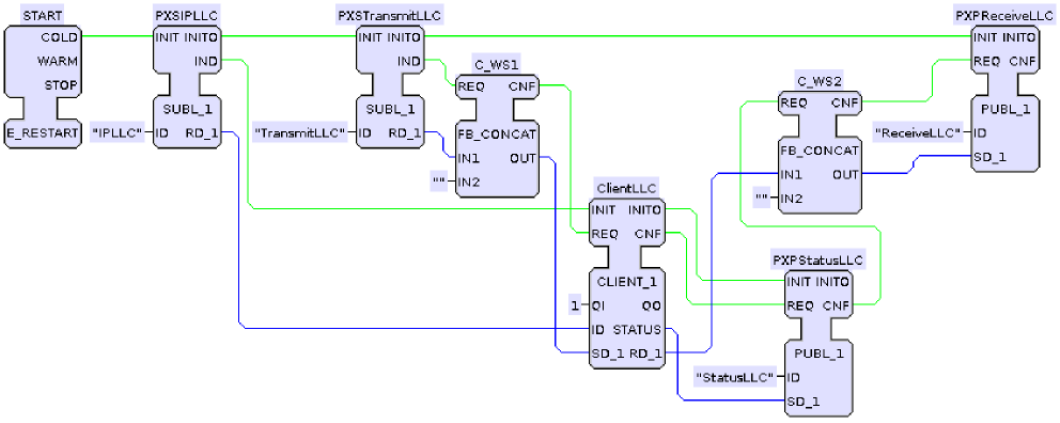


(c) The HL_Manual function block network

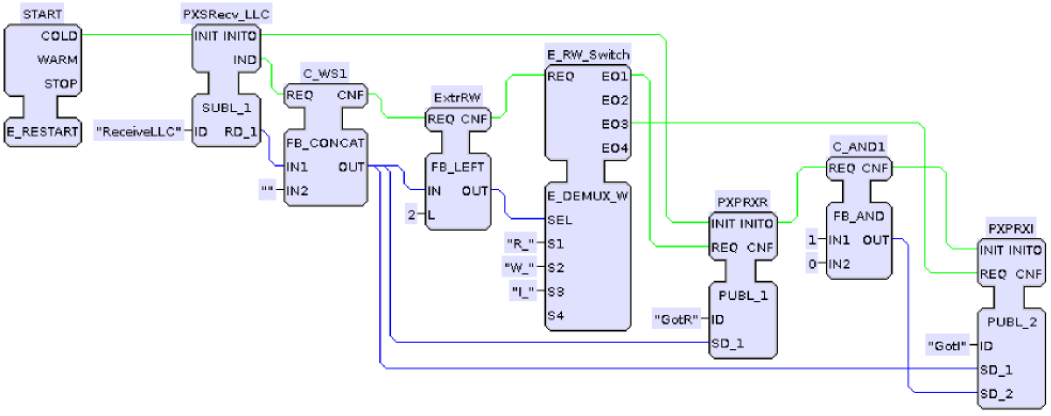


(d) The GetR function block network

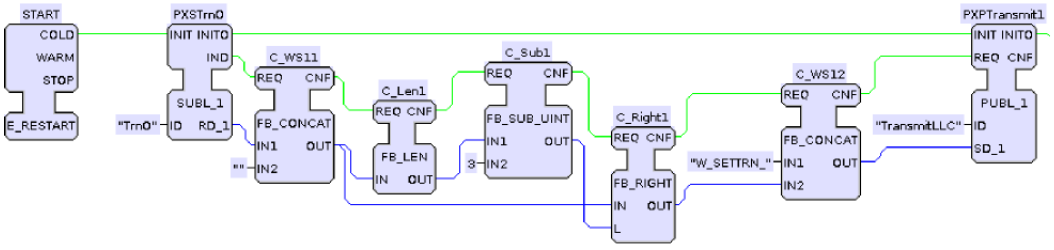
Figure B.2: Controller function block networks



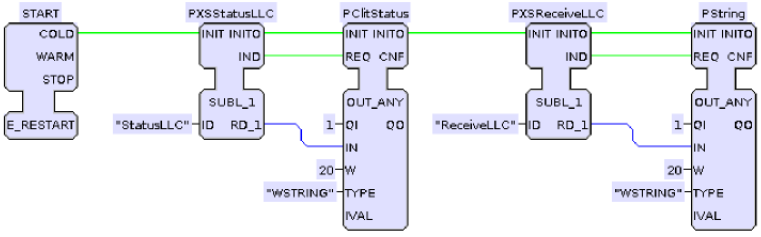
(a) The XSocket1 function block network



(b) The RX1 function block network



(c) The TX1 function block network



(d) The XInfo function block network

Figure B.3: Interface function block networks

E_Split1. One of the events from E_Split1 is sent to E_Perm1 that only permits the incoming event to pass through it if `PERMIT` (connected to C_PosInit's `On`) is true. E_Delay1 holds the incoming event for 10 milliseconds and then forward it to E_Merge1 that emits an event to repeat the process. Note that E_Merge1 emits an event if an event is received on any of its event inputs.

The other event from E_Split1 is sent to C_Post1 that concatenates "`PID`" and the counter number from `CI_ID`. C_Post2 further appends "`_PXIk`" to the result of C_Post1, resulting in a string `PIDxx_PXIk`, where `xx` is the counter value. PXIk publishes a value of zero from C_Int_0 to subscribers in the position blocks with IDs `PIDxx_PXIk`.

B.3 Event Splitting

The following paragraphs highlight an issue of function blocks, caused by splitting of events in event-based execution. Desc_Switch (a function block in figure B.2(d)) uses the output of C_Left to switch between either forwarding an event to PX_Pos or PX_Sens. A situation is encountered where, receiving either "`_0`" or "`_1`", emits an event of `EO1` or `EO2` on C_WS_E. Using these events, a boolean value must be set or cleared, followed by an event emitted to Desc_Switch. `EO1` and `EO2` on C_WS_E are each split by E_Split1 and E_Split2. The `EO1` outputs of E_Split1 and E_Split2 are sent to the `R` and `S` inputs of E_Bool, an E_RS type function block. This function block sets `Q` of E_Bool false for an event on `R`, and true for an event on `S`.

The `EO2` outputs of E_Split1 and E_Split2 are sent to E_Merge1. The event from E_Merge1, a new split from the E_Bool event line, is used to propagate to other function blocks. Ideally, `EO` of E_Bool should be used as the main event line to ensure the safe usage of `Q` on E_Bool, only when it is available. However, the standard E_RS does not emit an event on every incoming event, but only when `Q` has changed.

To ensure that Desc_Switch receive an event when E_RS is complete, an E_DELAY function block is used to hold the event for 10 milliseconds, assuming that E_RS will always be finished before other function blocks need its results. This is undesired since it breaks down the event based execution and wastes time. A short program code or a custom function block may solve this problem, but a weakness in programming with a standard set of function blocks seems to exist.

B.4 Interface Devices

The functioning of the interface devices are as follows. XSocket1 (figure B.3(a)) contains PXSIPLLC that subscribes to the address "`IPLLC`" that is published on by PXPIPLLC in resource ModeC of the HMI device. PXSIPLLC then

sends the address to ClientLLC's `ID` input. PXSTransmitLLC and PXPReceiveLLC subscribe and publish to TX1 and RX1 and are connected to SD_1 and RD_1 of ClientLLC.

PXSRecv_LLC in RX1 subscribes to ReceiveLLC and looks at the first 2 characters of the packet to determine whether to publish the packet on "`GotR`" (refer to figure B.1(d)) or "GotI". Figure B.2(c) shows a part that receives events, converts it to the packet format and publishes it to XSocket1. PXSTrnO subscribes to publishers in the FBC_Trans blocks that publish messages in the format "`TIDy`", on the address "`TrnO`". The transition ID (`y`) is then extracted and appended to "`W_SETTRN_`" and published on address "`TransmitLLC`". In the same way, other events are also received, processed and sent to XSocket1. Finally, figure B.2(d) shows the panel resource that subscribes to StatusLLC and ReceiveLLC and displays the information in text boxes.

Appendix C

Object-Orientated Controller Code

Code C.1 shows the main program code of the object-orientated controller. Code C.2 shows extra classes of the object-orientated controller.

Code C.1

```
// Declaration of the menu thread function
void *uithr(void *d);

struct HolonInfo{
  AMoveHolon *Holon;
  // Some other declarations
};

int main(int argc, char *argv[]){

  // Creating a request vector,
  // used to set the default state to enter from state p
  char *userRequest, *internalRequest, RequestM[8];
  userRequest = (RequestM + 4);
  internalRequest = RequestM;

  //Initialising the request vector
  for (f1 = 0; f1 < 8; f1++) RequestM[f1] = '@';
  RequestM[0] = 'j'; //default state

  // Creating a vector to hold the HolonInfo struct
  // that contains the task holons
  vector<HolonInfo> MoveHolonsVector;

  // Starting the FSM
  AStateMachine StateMachine1(internalRequest);
  if (StateMachine1.StartMachine()) return 1;
```

```
  else StateMachine1.StateX('a'); //Switch to state a

  // Starting menu thread
  int ui_thr_id;
  pthread_t ui_thr;
  ui_thr_id = pthread_create(&ui_thr,
      NULL, uithr, (void*) userRequest);

  // Starting the communication sockets
  AClient Client1;
  AClient Client2;
  if (StateMachine1.StateA() == 'a'){
    if (Client1.StartClient(argv[1],argv[2])) return 2;
    else if (Client2.StartClient(argv[1],argv[3])) return 2;
    else StateMachine1.StateX('b'); //Switch to state b
  }

  // Starting the configuration processor
  AConfigProcessor ConfigProcessor1(atoi(argv[4]),
      atof(argv[5]), atof(argv[6]));

  if (StateMachine1.StateA() == 'b'){
    if (ConfigProcessor1.StartProcessor()) return 3;
    else StateMachine1.StateX('p'); //Switch to state p
  }

  do{
    //The controller code
  } while (StateMachine1.StateA() != 'x');

  StateMachine1.StateX('y');
  *(userRequest + 3) = 'y';

  pthread_join(ui_thr, NULL); // Wait for menu thread to return
  StateMachine1.StateX('z');
  return 0;
}
```

Code C.2

```
class ADataTransfer {

  protected:
    //Some declarations
```

```
  public:
    virtual int FindChild(int, int *, short int) {return 0;}
    virtual int FindOtherOut(int) {return 0;}
    //Some other declarations
};

class AMoveHolon: public ADataTransfer {

  public:
    AMoveHolon(int, int, ADataTransfer *);
    ~AMoveHolon();
    int StartMoveHolon();
    int RequestTransition() {advance = 1; return 0;}
    int ServiceMoveHolon();

  private:
    ADataTransfer *ConfigProcessor1;
    //Some other declarations
};

class AConfigProcessor: public ADataTransfer {

  public:
    AConfigProcessor(int, float, float);
    ~AConfigProcessor();
    int StartProcessor();
    int FindChild(int, int *, short int);
    int FindOtherOut(int);
    //Some other declarations

  private:
    vector<vector<double>> biasTable;
    struct PosEntity {
      int ID;
      char name[64], type, domain[64];
    };
    struct TransEntity {
      int ID;
      char name[64], subName, domain[64];
      PosEntity *start, *end;
    };
    struct TransEntityComp {
      int ID;
      PosEntity *start, *end;
```

```
    };
    struct PosBias {
      PosEntity *position;
      float bias;
    };
    //Some other declarations
};
```

List of References

Adams, A.O. (2010). *Control of reconfigurable assembly system.* MEng Mechatronic Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Bellifemine, F.L., Caire, G. and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE.* Wiley Series in Agent Technology.

Bongard, J., Zykov, V. and Lipson, H. (2006). Resilient machines through continuous self-modeling. *SCIENCE*, vol. 314, pp. 1118 – 1121.

Brennan, R.W. (2007). Toward real-time distributed intelligent control: A survey of research themes and applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 5, pp. 744 –765.

Burger, J.N. (2009). *Reconfigurable Conveyor System and Pallet Magazine.* BEng Mechatronic Final Year Project, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Butler, G.F. and Corbin, M.J. (1991). Introduction to object-oriented simulation. In: *IEE Colloquium on Object-Oriented Simulation and Control*, pp. 1/1–1/3.

Carpanzano, E. and Jovane, F. (2007). Advanced automation solutions for future adaptive factories. *CIRP Annals - Manufacturing Technology*, vol. 56, no. 1, pp. 435–438.

Christensen, J.H. (1994). Holonic manufacturing systems: Initial architecture and standards directions. In: *Proceedings of the First European Conference on Holonic Manufacturing Systems.* Hannover, Germany.

Committee on Visionary Manufacturing Challenges (1998). *Visionary Manufacturing Challenges for 2020.* Commission on Engineering and Technical Systems, National Research Council, The National Academies Press.

Department of Defense (2001). *Systems Engineering Fundamentals.* Prepared by the defence acquisition university press, Fort Belvoir, Virginia.

Diego, B.C., Rodilla, V.M., Carames, C.F., Moran, A.C. and Santos, R.A. (2010). Applying a software framework for supervisory control of a PLC-based flexible manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, vol. 48, pp. 663–669.

Du Preez, J. (2011). *A study of reconfigurable manufacturing systems with computer simulation.* MEng Mechatronic Thesis, Department of Industrial Engineering, Stellenbosch University.

Dymond, F.S.D. (2009). *Conceptual design of a fixtureless reconfigurable automated assembly system.* MEng Mechatronic Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Koestler, A. (1969). *The Ghost in the Machine.* Arkana Books, London.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. and Van Brussel, H. (1999). Reconfigurable manufacturing systems. *CIRP Annals - Manufacturing Technology*, vol. 48, no. 2, pp. 527–540.

Koren, Y. and Shpitalni, M. (2010). Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130–141.

Kruger, K. (2011). *Control of the feeder for a reconfigurable assembly system.* MEng Mechatronic Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Leitão, P., Barbosa, J. and Trentesaux, D. (2012). Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence*, vol. 25, no. 5, pp. 934–944.

Leitão, P. and Restivo, F. (2006). Adacor: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, vol. 57, no. 2, pp. 121 – 130.

Lepuschitz, W., Zoitl, A., Vallée, M. and Merdan, M. (2011). Toward self-reconfiguration of manufacturing systems using automation agents. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 52 –69.

Mcfarlane, D.C. and Bussmann, S. (2000). Developments in holonic production planning and control. *Production Planning & Control*, vol. 11, no. 6, pp. 522–536.

Meng, F., Tan, D. and Wang, Y. (2006). Development of agent for reconfigurable assembly system with jade. In: *Proceedings of the 6th World Congress on Intelligent Control and Automation.* Dalian, China.

Mulubika, C. (2013). *Evaluation of control strategies for reconfigurable manufacturing systems.* MEng Mechatronic Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Murata, T. (1989). Petri-nets: Properties, analysis and applications. *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541 –580.

Poletti, R. (2011). *Mechanical Design of a Stepped-Conveyor Singulation Unit, an internal design report.* Internal design report, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Sequeira, M.A. (2009). *Conceptual design of a fixture-based reconfigurable spot welding system.* MEng Mechatronic Thesis, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Strauss, R. (2009). *Development of a Reconfigurable Parts Feeder for Assembly Automation.* BEng Mechatronic Final Year Project, Department of Mechanical and Mechatronic Engineering, Stellenbosch University.

Valckenaers, P., Van Brussel, H., Wyns, J., Bongaerts, L. and Peeters, P. (1998). Designing holonic manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, vol. 14, no. 5, pp. 455–464.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P. (1998). Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, vol. 37, no. 3, pp. 255–274.

Vrba, P., Tichý, P., Mařík, V., Hall, K., Staron, R., Maturana, F. and Kadera, P. (2011). Rockwell automation's holonic and multiagent control systems compendium. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 14 –30.

Vyatkin, V. (2007). *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design.* ISA-Instrumentation, Systems, and Automation Society.

Wu, N. and Zhou, M. (2011). Intelligent token petri nets for modelling and control of reconfigurable automated manufacturing systems with dynamical changes. *Transactions of the Institute of Measurement and Control*, vol. 33, no. 1, pp. 9–29.

Zhang, J., Gu, J., Li, P. and Duan, Z. (1999). Object-oriented modeling of control system for agile manufacturing cells. *International Journal of Production Economics*, vol. 62, no. 1-2, pp. 145 – 153.